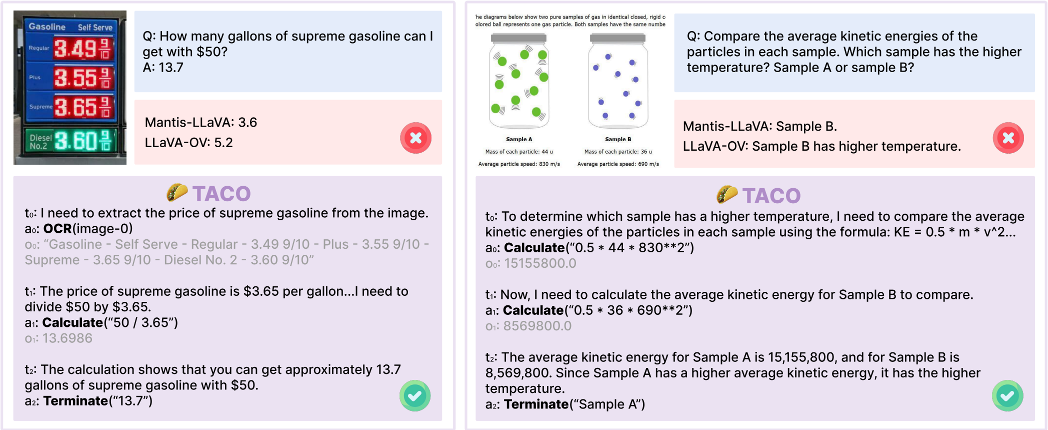
TACO: LEARNING MULTI-MODAL MODELS TO REASON AND ACT WITH SYNTHETIC DATA

Zixian Ma[1,2]*, **Jianguo Zhang**[2], **Zhiwei Liu**[2], **Jieyu Zhang**[1], **Juntao Tan**[2], **Manli Shu**[2],
Juan Carlos Niebles[2], **Shelby Heinecke**[2], **Huan Wang**[2],
Caiming Xiong[2], **Ranjay Krishna**[1], **Silvio Savarese**[2]
[1]University of Washington, [2]Salesforce Research
Website: `https://taco-project.github.io`

ABSTRACT

While open-source multi-modal language models perform well on simple question answering tasks, they often fail on complex questions that require multiple capabilities, such as fine-grained recognition, visual grounding, and reasoning, and that demand multi-step solutions. We present TACO, a family of multi-modal large action models designed to improve performance on such complex, multi-step and multi-modal tasks. During inference, TACO produces chains-of-thought-and–action (CoTA), executes intermediate steps by invoking external tools such as OCR, depth estimation and calculator, then integrates both the thoughts and action outputs to produce coherent responses. To train TACO, we create a large dataset of 1M+ synthetic CoTA traces generated with GPT-4o and Python programs. We then experiment with various data filtering and mixing techniques and obtain a final subset of 293K high-quality CoTA examples. This dataset enables TACO to learn complex reasoning and action paths, surpassing existing models trained on instruct tuning data with only direct answers. Our model TACO outperforms the instruction-tuned baseline across 8 benchmarks, achieving a 3.9% improvement on average, with gains up to 20% in MMVet tasks involving OCR, mathematical reasoning and spatial reasoning.


Gasoline
Self Serve
Regular
3.49 9/10
Plus
3.55 9/10
Supreme
3.65 9/10
Diesel No.2
3.60 9/10
TACO
TACO
he diagrams below show two pure samples of gas in identical closed, rigid c
olored ball represents one gas particle. Both samples have the same numbe
Sample A
Mass of each particle: 44 u
Average particle speed: 830 m/s
Sample B
Mass of each particle: 36 u
Average particle speed: 690 m/s

Q: How many gallons of supreme gasoline can I get with $50?
A: 13.7

Mantis-LLaVA: 3.6
LLaVA-OV: 5.2

t_0: I need to extract the price of supreme gasoline from the image.
a_0: **OCR**(image-0)
o_0: "Gasoline - Self Serve - Regular - 3.49 9/10 - Plus - 3.55 9/10 - Supreme - 3.65 9/10 - Diesel No. 2 - 3.60 9/10"

t_1: The price of supreme gasoline is $3.65 per gallon...I need to divide $50 by $3.65.
a_1: **Calculate**("50 / 3.65")
o_1: 13.6986

t_2: The calculation shows that you can get approximately 13.7 gallons of supreme gasoline with $50.
a_2: **Terminate**("13.7")

Q: Compare the average kinetic energies of the particles in each sample. Which sample has the higher temperature? Sample A or sample B?

Mantis-LLaVA: Sample B.
LLaVA-OV: Sample B has higher temperature.

t_0: To determine which sample has a higher temperature, I need to compare the average kinetic energies of the particles in each sample using the formula: KE = 0.5 * m * v^2...
a_0: **Calculate**("0.5 * 44 * 830**2")
o_0: 15155800.0

t_1: Now, I need to calculate the average kinetic energy for Sample B to compare.
a_1: **Calculate**("0.5 * 36 * 690**2")
o_1: 8569800.0

t_2: The average kinetic energy for Sample A is 15,155,800, and for Sample B is 8,569,800. Since Sample A has a higher average kinetic energy, it has the higher temperature.
a_2: **Terminate**("Sample A")

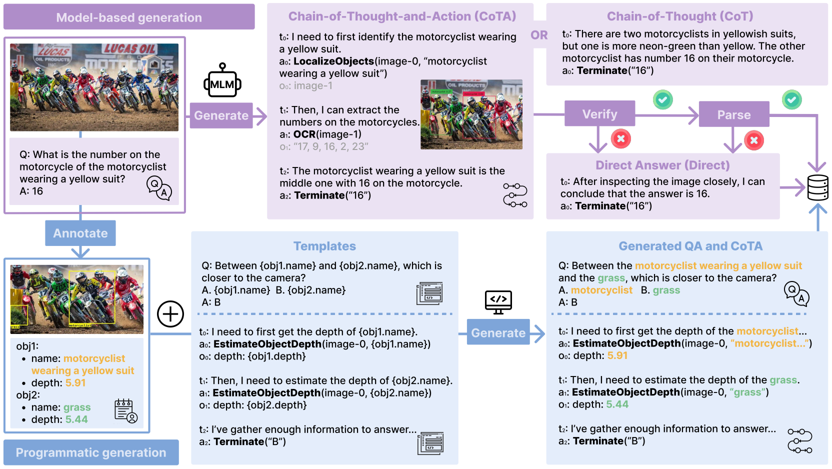
Figure 1: **Example outputs of TACO vs. other multi-modal large language models.** Our TACO model is able to output a Chain-of-Thought-and-Action (CoTA) and answer challenging questions based on the thoughts and action outputs, whereas existing multi-modal large language models fail.

1 INTRODUCTION

The space of real-world multi-modal problems is enormous, ranging from simple question answering Antol et al. (2015) to difficult multi-step mathematical reasoning Hu et al. (2024a). For example, an user can take a photo of a panel with different gasoline prices and ask an agent how much gasoline they can buy with a certain budget (Figure 1). While existing multi-modal language models excel at answering straightforward questions, they still struggle at ones like this, which requires localization and OCR to find gasoline types and prices, and multi-step reasoning and calculation to respond. When models fail, it is hard to diagnose which implicit step caused the error. Worse, all

---

[*] Work done while interning at Salesforce Research



Q: What is the number on the motorcycle of the motorcyclist wearing a yellow suit?
A: 16

t_0: I need to first identify the motorcyclist wearing a yellow suit.
a_0: **LocalizeObjects**(image-0, "motorcyclist wearing a yellow suit")
o_0: image-1

t_1: Then, I can extract the numbers on the motorcycles.
a_1: **OCR**(image-1)
o_1: "17, 9, 16, 2, 23"

t_2: The motorcyclist wearing a yellow suit is the middle one with 16 on the motorcycle.
a_2: **Terminate**("16")

t_0: There are two motorcyclists in yellowish suits, but one is more neon-green than yellow. The other motorcyclist has number 16 on their motorcycle.
a_0: **Terminate**("16")

t_0: After inspecting the image closely, I can conclude that the answer is 16.
a_0: **Terminate**("16")

obj1:
- name: motorcyclist wearing a yellow suit
- depth: 5.91

obj2:
- name: grass
- depth: 5.44

Q: Between {obj1.name} and {obj2.name}, which is closer to the camera?
A. {obj1.name} B. {obj2.name}
A: B

t_0: I need to first get the depth of {obj1.name}.
a_0: **EstimateObjectDepth**(image-0, {obj1.name})
o_0: depth: {obj1.depth}

t_1: Then, I need to estimate the depth of {obj2.name}.
a_1: **EstimateObjectDepth**(image-0, {obj2.name})
o_1: depth: {obj2.depth}

t_2: I've gather enough information to answer...
a_2: **Terminate**("B")

Q: Between the motorcyclist wearing a yellow suit and the grass, which is closer to the camera?
A. motorcyclist B. grass
A: B

t_0: I need to first get the depth of the motorcyclist...
a_0: **EstimateObjectDepth**(image-0, "motorcyclist...")
o_0: depth: 5.91

t_1: Then, I need to estimate the depth of the grass.
a_1: **EstimateObjectDepth**(image-0, "grass")
o_1: depth: 5.44

t_2: I've gather enough information to answer...
a_2: **Terminate**("B")

Figure 2: **Data generation.** We illustrate our model-based data generation (top) and programmatic generation (bottom) pipelines.

open-source multi-modal models struggle to outline the steps to solve complex problems because their instruction-tuning datasets are dominated with single-step solutions and brief direct answers.

However, there is a promising and efficient alternative: multi-modal large action models Hu et al. (2024a); Gupta & Kembhavi (2022); Surís et al. (2023); Hu et al. (2024b). Multi-modal action models can, at inference time, perform actions by invoking external tools, which can be specialized vision models (e.g., object detection and OCR), API calls (e.g., web search), or any other Python functions (e.g., sorting) Hu et al. (2024a); Surís et al. (2023). They explicitly reason and execute actions, generating a *chain-of-thought-and-action* (CoTA) needed to produce the final answer. It has been shown that even proprietary frontier models, such as GPT-4o, improve on mathematical and vision-centric tasks when augmented with CoTA Hu et al. (2024a). Unlike GPT-4o, open-source multi-modal models fail at action calling without finetuning Hu et al. (2024b); Liu et al. (2023c); Qi et al. (2024); Shao et al. (2024). This direction is still in its infancy; existing methods have not introduced large-scale or high-quality CoTA data Liu et al. (2023c), or experimented with more than half-a-dozen tools Qi et al. (2024). To train effective multi-modal action models, we need high-quality instruction-tuning data in the CoTA format to help models learn to call diverse actions and perform complex reasoning, which are expensive to obtain at scale with human annotators.

To achieve this goal with a reasonable budget, we turn to synthetic data generation instead. We leverage both GPT-4o and Python programs to generate large-scale synthetic CoTA data across diverse image sources, which we filter and mix aggressively using various techniques. In total, we have generated 815K CoTA traces with GPT-4o across 31 datasets and 1M+ traces with handcrafted programs. With these data, we compose 9 different recipes and perform extensive experiments with 3 open-source multi-modal models of different language models (i.e. Qwen2 and LLaMA3) and vision encoders (i.e. CLIP and SigLIP). Finally, we identify the optimal recipe and obtain a final subset of 293K high-quality CoTA data, which we then use to train and release a family of multi-modal action models named TACO.

We evaluate our models on 8 benchmarks covering diverse capabilities including general visual question answering, finegrained object and optical character recognition, math and multi-step reasoning, as well as multi-image understanding. Our best TACO performs on average 3.9% better across all benchmarks than the baseline trained with instruction-tuning data with only direct answers. Notably, we observe a staggering $10-20\%$ improvement on MMVet, which tests for diverse capabilities with a focus on OCR and mathematical reasoning. Training on high-quality CoTA traces sets a new standard for complex multi-modal reasoning, highlighting the need for structured, multi-step instruction tuning in advancing open-source mutli-modal models' capabilities.

2 CHAIN-OF-THOUGHT-AND-ACTION (COTA)

Our goal is to generate synthetic chains-of-thought-and-action and use them to train multi-modal action models. We define a chain-of-thought-and-action C as a sequence of steps S_i, where each

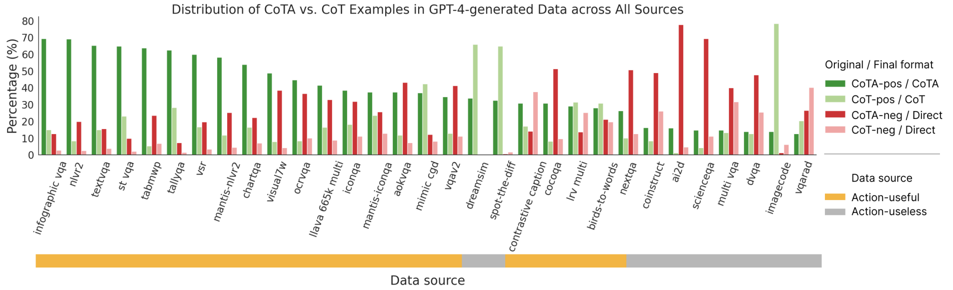


Figure 3: **Data distribution.** We visualize the frequency of the final data format (i.e. CoTA, CoT, or Direct) in each dataset across all data sources. We also highlight the Action-useless (i.e. % of CoT-pos − CoTA-pos > 10 or % of CoTA-neg − CoTA-pos > 10) vs. Action-useful datasets.

step consists of thought t_i, action a_i and observation o_i:

$$C = (S_0, S_1, ..., S_n) = (S_i)_{i=0}^{n} \tag{1}$$

$$S_i = (t_i, a_i, o_i), t_i \in L, a_i \in A \tag{2}$$

where L represents the space of language, and A is the action space as described next. Note that the model only generates t_i and a_i, where the training loss is applied on, whereas o_i is obtained by executing a_i in Python code.

2.1 DATA GENERATION

Due to the prohibitive cost of obtaining manually constructing CoTA traces at scale, we generate synthetic CoTA traces with two automatic approaches: Model-based and programmatic generation.

Model-based generation. Inspired by prior work that demonstrates the utility of model-generated data Hu et al. (2023), we design a model-based data generation pipeline (Figure 2 top):

1. **GENERATE**. First, we leverage images and QA examples in existing visual instruction tuning datasets and generate chain-of-thought-and-action traces to solve the questions with GPT-4o. We include diverse questions on both single-image and multi-image examples from two large-scale instruction tuning datasets Cauldron and Mantis-Instruct Jiang et al. (2024); Laurençon et al. (2024). We feed the images and questions to GPT-4o and prompt it to answer the questions by outputting a CoTA or just a CoT without actions when actions are not necessary or helpful (Figure 2).

2. **VERIFY**. Second, we verify GPT-4o's generated answers against the groundtruth in the instruction tuning datasets. We force GPT-4o to always end with TERMINATE(answer) and compare its predicted answer to the groundtruth. If the final answer is incorrect, we convert this CoTA-neg example into the direct answer (Direct) format with the correct answer (Figure 2).

3. **PARSE**. Finally, we run `json.loads(`S_i`)` on each step of the CoTA traces with correct final answers. Similar to the previous stage, we again keep the CoTA examples that are parsed successfully and turn the failed examples into the Direct format with groundtruth answers.

Programmatic data generation. We also explore a more cost-efficient and controllable method: programmatic data generation (Figure 2 bottom). This pipeline involves two main steps:

1. **ANNOTATE**. First, we gather existing dense annotations of images. To this end, we leverage Visual Genome (VG) as it contains rich human annotations of objects, attributes, and relationships in the images. In addition, we also obtain depth maps of the VG images with Depth-Anything-v2 Yang et al. (2024) as depths are useful for many visual tasks but hard to manually annotate.

2. **GENERATE**. Next, based on the dense annotations of images, we programmatically generate both the QA pairs and the corresponding CoTA traces with manually written templates. We reuse the pipeline from Zhang et al. (2024b;c) for generating diverse QA pairs that cover various vision capabilities such as counting and spatial understanding (See Appendix for more details).

In total, we generate 815K CoTA traces with GPT-4o, and 1M+ QA pairs and corresponding CoTA data with programs. We then develop 9 different data recipes with 3 filtering/mixing techniques, where we vary the distribution of (1) data formats (Figure 3), (2) data sources (Figure 3), and (3) model-generated vs. program-generated data.

3 EXPERIMENTS

We perform extensive experiments with 3 open-source multi-modal models and 9 data recipes on 8 benchmarks to study the effectiveness of CoTA data compared to instruction-tuning data with only direct answers and investigate whether filtering and programmatic data can lead to further gains.

Table 1: **CoTA inference before vs. after fine-tuning.** We show that fine-tuning with CoTA data elicits their reasoning and action calling abilities and significantly boosts their performance.

Model	Language / Vision backbone	Train data / Inference format	A-OKVQA	BLINK	MathVista	MMMU	MMStar	MMVet	MMVP	RealWorldQA	Avg
GPT-4o (2024-08-06)	—	— / Direct	88.4	64.7	60.5	67.6	64.5	70.0	84.7	72.0	71.5
		— / CoTA	89.9	63.2	59.0	64.6	64.3	67.2	83.0	69.9	70.1
GPT-4o (language-only)		— / CoTA	74.8	45.6	44.5	54.1	55.3	45.2	58.0	50.2	53.5
Mantis-instruction-tuned	LLaMA3-8B / SigLIP	— / Direct	81.2	46.4	34.4	40.1	40.1	36.9	69.0	51.0	49.9
		— / CoTA	0.5	0.0	20.0	1.5	1.7	0.0	0.0	0.0	3.0
TACO		CoTA 293K / CoTA	81.8	47.6	36.3	40.9	42.5	45.7	65.3	56.5	52.1
LLaVA-OV-Stage1.5	Qwen2-7B / SigLIP	— / Direct	76.1	34.8	35.9	36.1	39.1	32.3	63.7	54.1	46.5
		— / CoTA	25.7	8.8	21.5	21.2	26.7	7.2	40.5	37.5	23.6
TACO		CoTA 293K / CoTA	**86.1**	**52.6**	**46.9**	**44.0**	**50.8**	**51.2**	**73.7**	**61.1**	**58.3**

Table 2: **Best CoTA data recipe.** CoTA data improves models' average performance across 8 multi-modal benchmarks by 1-4%. We highlight whether CoTA data increases or decreases performance on a particular benchmark compared to the baseline.

Model	Language / Vision backbone	Start checkpoint / Seen data	Train data / Inference format	A-OKVQA	BLINK	MathVista	MMMU	MMStar	MMVet	MMVP	RealWorldQA	Avg	Delta
Mantis	LLaMA3-8B / CLIP	Pretrained / 558K	Direct 293K / Direct	80.7	45.8	33.1	42.2	36.7	28.9	62.7	52.3	47.8	
TACO			CoTA 293K / CoTA	81.1	49.6	36.6	42.8	40.8	45.2	63.3	51.1	51.3	+3.5
Mantis	LLaMA3-8B / SigLIP	Pretrained / 558K	Direct 293K / Direct	80.3	43.7	31.1	40.4	40.5	33.0	63.3	51.8	48.0	
TACO			CoTA 293K / CoTA	82.4	47.8	34.9	40.3	44.6	45.2	64.0	53.7	51.6	+3.6
Mantis		Instruction tuned / 1.2M	Direct 293K / Direct	81.1	46.7	36.2	40.7	40.7	29.7	68.3	54.8	49.8	
TACO			CoTA 293K / CoTA	82.8	47.3	38.9	41.9	45.1	50.0	67.7	56.1	53.7	**+3.9**
LLaVA-OV	Qwen2-7B / SigLIP	Stage 1 / 558K	Direct 293K / Direct	83.1	49.5	38.4	45.6	42.3	33.0	69.7	55.3	52.1	
TACO			CoTA 293K / CoTA	84.5	49.6	41.8	45.3	44.5	48.9	66.7	53.6	54.4	+2.3
LLaVA-OV		Stage 1.5 / 4.5M	Direct 293K / Direct	85.5	50.3	42.4	**46.1**	50.1	39.3	73.6	57.8	55.6	
TACO			CoTA 293K / CoTA	**86.1**	**52.6**	**46.9**	44.0	**50.8**	**51.2**	**73.7**	**61.1**	**58.3**	+2.5

Models and baselines. We experiment with three models with different language models and vision encoders: Mantis-8B-SigLIP (based on LLaMA3-8B and SigLIP), Mantis-8B-CLIP (LLaMA3-8B and CLIP), and LLaVA-OneVision-7B (Qwen2-7B and SigLIP). We train our baselines with the same hyperparameters and the same examples in the Direct instead of the CoTA format.

Evaluation setup. We select 8 multi-modal benchmarks covering diverse capabilities, with a focus on vision-centric question answering benchmarks, including MMVet, MMVP, RealWorldQA, A-OKVQA, and BLINK Yu et al. (2024); Tong et al. (2024); Schwenk et al. (2022); Fu et al. (2024); Li et al. (2023b). Additionally, we have also included one visual mathematical reasoning benchmark MathVista and two general multi-modal benchmarks MMMU and MMStar Lu et al. (2024a); Chen et al. (2024a); Yue et al. (2024). We adapt VLMEvalKit Duan et al. (2024) for our evaluation. Additional evaluation details can be found in the Appendix.

3.1 MAIN RESULTS

Fine-tuning open-source multi-modal models with CoTA data elicits their reasoning and action calling abilities. As in Hu et al. (2024a), we observe that open-source multi-modal models fail to call external actions with few-shot prompting, even though proprietary model GPT-4o does it well (Table 1). However, fine-tuning them with our CoTA data effectively elicits their abilities to produce chains of thoughts and actions at inference time (Table 1).

Our best CoTA data recipe results in a strong multi-modal action model TACO that consistently beats instruction-tuned baselines by 1-4% on average across 8 benchmarks, with significant gains on MMVet. We have fine-tuned 3 different multi-modal models with our best data recipe CoTA 293K starting from different checkpoints. We observe that our CoTA data leads to consistent gains in the model's average accuracy across 8 benchmarks compared to the instruction-tuned baselines trained with the same examples in the Direct format (Table 2). Our CoTA data results in larger gains on some benchmarks than others: we see 10-20% gains on MMVet and 1-3% increases on other benchmarks except for MMMU and MMVP, which can experience up to 2-3% drops. We perform qualitative analysis on the greatest gains and drops and present examples in Section B.3. Additional results from ablation experiments can be found in Section B.1.

4 CONCLUSION

We generate large-scale synthetic Chain-of-Thought-and-Action data and perform extensive experiments with 9 data recipes and 3 open-source multi-modal models across 8 benchmarks. We demonstrate quantitatively and qualitatively, enhanced capabilities of our multi-modal large action model TACO compared to baselines instruction-tuned with only direct answers.

REFERENCES

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pp. 2425–2433, 2015.

Anas Awadalla, Irena Gao, Josh Gardner, Jack Hessel, Yusuf Hanafy, Wanrong Zhu, Kalyani Marathe, Yonatan Bitton, Samir Gadre, Shiori Sagawa, Jenia Jitsev, Simon Kornblith, Pang Wei Koh, Gabriel Ilharco, Mitchell Wortsman, and Ludwig Schmidt. Openflamingo: An open-source framework for training large autoregressive vision-language models, 2023.

Jing Bi, Nguyen Manh Nguyen, Ali Vosoughi, and Chenliang Xu. Misar: A multimodal instructional system with augmented reality, 2023.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning, 2023a. URL `https://arxiv.org/abs/2310.05915`.

Guo Chen, Yin-Dong Zheng, Jiahao Wang, Jilan Xu, Yifei Huang, Junting Pan, Yi Wang, Yali Wang, Yu Qiao, Tong Lu, and Limin Wang. Videollm: Modeling video sequence with large language models, 2023b.

Houlun Chen, Xin Wang, Hong Chen, Zihan Song, Jia Jia, and Wenwu Zhu. Grounding-prompter: Prompting llm with multimodal information for temporal sentence grounding in long videos, 2023c.

Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, et al. Are we on the right way for evaluating large vision-language models? *arXiv preprint arXiv:2403.20330*, 2024a.

Xi Chen, Xiao Wang, Lucas Beyer, Alexander Kolesnikov, Jialin Wu, Paul Voigtlaender, Basil Mustafa, Sebastian Goodman, Ibrahim Alabdulmohsin, Piotr Padlewski, Daniel Salz, Xi Xiong, Daniel Vlasic, Filip Pavetic, Keran Rong, Tianli Yu, Daniel Keysers, Xiaohua Zhai, and Radu Soricut. Pali-3 vision language models: Smaller, faster, stronger, 2023d.

Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, Bin Li, Ping Luo, Tong Lu, Yu Qiao, and Jifeng Dai. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. *arXiv preprint arXiv:2312.14238*, 2023e.

Zhe Chen, Weiyun Wang, Hao Tian, Shenglong Ye, Zhangwei Gao, Erfei Cui, Wenwen Tong, Kongzhi Hu, Jiapeng Luo, Zheng Ma, et al. How far are we to gpt-4v? closing the gap to commercial multimodal models with open-source suites. *arXiv preprint arXiv:2404.16821*, 2024b.

Zhipeng Chen, Kun Zhou, Beichen Zhang, Zheng Gong, Wayne Xin Zhao, and Ji-Rong Wen. Chatcot: Tool-augmented chain-of-thought reasoning on chat-based large language models. *arXiv preprint arXiv:2305.14323*, 2023f.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale N Fung, and Steven Hoi. Instructblip: Towards general-purpose vision-language models with instruction tuning. *Advances in Neural Information Processing Systems*, 36, 2024.

Haodong Duan, Junming Yang, Yuxuan Qiao, Xinyu Fang, Lin Chen, Yuan Liu, Xiaoyi Dong, Yuhang Zang, Pan Zhang, Jiaqi Wang, Dahua Lin, and Kai Chen. Vlmevalkit: An open-source toolkit for evaluating large multi-modality models, 2024. URL `https://arxiv.org/abs/2407.11691`.

Xingyu Fu, Yushi Hu, Bangzheng Li, Yu Feng, Haoyu Wang, Xudong Lin, Dan Roth, Noah A Smith, Wei-Chiu Ma, and Ranjay Krishna. Blink: Multimodal large language models can see but not perceive. *arXiv preprint arXiv:2404.12390*, 2024.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, pp. 10764–10799. PMLR, 2023.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023.

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. *ArXiv*, abs/2211.11559, 2022.

Joy He-Yueya, Gabriel Poesia, Rose E Wang, and Noah D Goodman. Solving math word problems by combining language models with symbolic solvers. *arXiv preprint arXiv:2304.09102*, 2023.

Yushi Hu, Otilia Stretcu, Chun-Ta Lu, Krishnamurthy Viswanathan, Kenji Hata, Enming Luo, Ranjay Krishna, and Ariel Fuxman. Visual program distillation: Distilling tools and programmatic reasoning into vision-language models, 2023.

Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, and Ranjay Krishna. Visual sketchpad: Sketching as a visual chain of thought for multimodal language models, 2024a. URL `https://arxiv.org/abs/2406.09403`.

Yushi Hu, Otilia Stretcu, Chun-Ta Lu, Krishnamurthy Viswanathan, Kenji Hata, Enming Luo, Ranjay Krishna, and Ariel Fuxman. Visual program distillation: Distilling tools and programmatic reasoning into vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9590–9601, 2024b.

Bin Huang, Xin Wang, Hong Chen, Zihan Song, and Wenwu Zhu. Vtimellm: Empower llm to grasp video moments. *arXiv preprint arXiv:2311.18445*, 2(3):9, 2023.

Dongfu Jiang, Xuan He, Huaye Zeng, Con Wei, Max Ku, Qian Liu, and Wenhu Chen. Mantis: Interleaved multi-image instruction tuning. *arXiv preprint arXiv:2405.01483*, 2024.

Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Fei-Fei Li. Visual genome: Connecting language and vision using crowdsourced dense image annotations, 2016. URL `https://arxiv.org/abs/1602.07332`.

Hugo Laurençon, Léo Tronchon, Matthieu Cord, and Victor Sanh. What matters when building vision-language models?, 2024.

Bo Li, Yuanhan Zhang, Liangyu Chen, Jinghao Wang, Fanyi Pu, Jingkang Yang, Chunyuan Li, and Ziwei Liu. Mimic-it: Multi-modal in-context instruction tuning, 2023a.

Bo Li, Yuanhan Zhang, Dong Guo, Renrui Zhang, Feng Li, Hao Zhang, Kaichen Zhang, Yanwei Li, Ziwei Liu, and Chunyuan Li. Llava-onevision: Easy visual task transfer. *arXiv preprint arXiv:2408.03326*, 2024.

Bohao Li, Yuying Ge, Yixiao Ge, Guangzhi Wang, Rui Wang, Ruimao Zhang, and Ying Shan. Seed-bench-2: Benchmarking multimodal large language models. *arXiv preprint arXiv:2311.17092*, 2023b.

Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *ICML*, 2022.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. BLIP-2: bootstrapping language-image pre-training with frozen image encoders and large language models. In *ICML*, 2023c.

Kevin Lin, Faisal Ahmed, Linjie Li, Chung-Ching Lin, Ehsan Azarnasab, Zhengyuan Yang, Jianfeng Wang, Lin Liang, Zicheng Liu, Yumao Lu, Ce Liu, and Lijuan Wang. Mm-vid: Advancing video understanding with gpt-4v(ision), 2023.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning, 2023a.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning, 2023b.

Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. Llava-next: Improved reasoning, ocr, and world knowledge, January 2024a. URL `https://llava-vl.github.io/blog/2024-01-30-llava-next/`.

Shikun Liu, Linxi Fan, Edward Johns, Zhiding Yu, Chaowei Xiao, and Anima Anandkumar. Prismer: A vision-language model with multi-task experts, 2024b.

Shilong Liu, Hao Cheng, Haotian Liu, Hao Zhang, Feng Li, Tianhe Ren, Xueyan Zou, Jianwei Yang, Hang Su, Jun Zhu, Lei Zhang, Jianfeng Gao, and Chunyuan Li. Llava-plus: Learning to use tools for creating multimodal agents, 2023c. URL `https://arxiv.org/abs/2311.05437`.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models, 2023.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. In *International Conference on Learning Representations (ICLR)*, 2024a.

Pan Lu, Baolin Peng, Hao Cheng, Michel Galley, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, and Jianfeng Gao. Chameleon: Plug-and-play compositional reasoning with large language models. *Advances in Neural Information Processing Systems*, 36, 2024b.

Chenyang Lyu, Minghao Wu, Longyue Wang, Xinting Huang, Bingshuai Liu, Zefeng Du, Shuming Shi, and Zhaopeng Tu. Macaw-llm: Multi-modal language modeling with image, audio, video, and text integration, 2023.

Zixian Ma, Weikai Huang, Jieyu Zhang, Tanmay Gupta, and Ranjay Krishna. m&m's: A benchmark to evaluate tool-use for multi-step multi-modal tasks. *EECV 2024*, 2024.

Michael Moor, Qian Huang, Shirley Wu, Michihiro Yasunaga, Cyril Zakka, Yash Dalmia, Eduardo Pontes Reis, Pranav Rajpurkar, and Jure Leskovec. Med-flamingo: a multimodal medical few-shot learner, 2023.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.

Artemis Panagopoulou, Le Xue, Ning Yu, Junnan Li, Dongxu Li, Shafiq R. Joty, Ran Xu, Silvio Savarese, Caiming Xiong, and Juan Carlos Niebles. X-instructblip: A framework for aligning x-modal instruction-aware representations to llms and emergent cross-modal reasoning. *ArXiv*, abs/2311.18799, 2023. URL `https://api.semanticscholar.org/CorpusID:265506093`.

Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. Art: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*, 2023.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023.

Zhiliang Peng, Wenhui Wang, Li Dong, Yaru Hao, Shaohan Huang, Shuming Ma, and Furu Wei. Kosmos-2: Grounding multimodal large language models to the world, 2023.

Ji Qi, Ming Ding, Weihan Wang, Yushi Bai, Qingsong Lv, Wenyi Hong, Bin Xu, Lei Hou, Juanzi Li, Yuxiao Dong, and Jie Tang. Cogcom: Train large vision-language models diving into details through chain of manipulations, 2024. URL `https://arxiv.org/abs/2402.04236`.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023. URL `https://arxiv.org/abs/2307.16789`.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Tool learning with large language models: A survey. *arXiv preprint arXiv:2405.17935*, 2024.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools, 2023. URL `https://arxiv.org/abs/2302.04761`.

Dustin Schwenk, Apoorv Khandelwal, Christopher Clark, Kenneth Marino, and Roozbeh Mottaghi. A-okvqa: A benchmark for visual question answering using world knowledge, 2022. URL `https://arxiv.org/abs/2206.01718`.

Hao Shao, Shengju Qian, Han Xiao, Guanglu Song, Zhuofan Zong, Letian Wang, Yu Liu, and Hongsheng Li. Visual cot: Unleashing chain-of-thought reasoning in multi-modal language models, 2024.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36, 2024.

Mustafa Shukor, Corentin Dancette, Alexandre Rame, and Matthieu Cord. Unival: Unified model for image, video, audio and language tasks, 2023.

Guangzhi Sun, Wenyi Yu, Changli Tang, Xianzhao Chen, Tian Tan, Wei Li, Lu Lu, Zejun Ma, and Chao Zhang. Fine-grained audio-visual joint representations for multimodal large language models, 2023.

Quan Sun, Yufeng Cui, Xiaosong Zhang, Fan Zhang, Qiying Yu, Zhengxiong Luo, Yueze Wang, Yongming Rao, Jingjing Liu, Tiejun Huang, and Xinlong Wang. Generative multimodal models are in-context learners, 2024a.

Quan Sun, Qiying Yu, Yufeng Cui, Fan Zhang, Xiaosong Zhang, Yueze Wang, Hongcheng Gao, Jingjing Liu, Tiejun Huang, and Xinlong Wang. Emu: Generative pretraining in multimodality, 2024b.

Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. *arXiv preprint arXiv:2303.08128*, 2023.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. Toolalpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*, 2023a.

Yunlong Tang, Jinrui Zhang, Xiangchen Wang, Teng Wang, and Feng Zheng. Llmva-gebc: Large language model with video adapter for generic event boundary captioning, 2023b.

Shengbang Tong, Zhuang Liu, Yuexiang Zhai, Yi Ma, Yann LeCun, and Saining Xie. Eyes wide shut? exploring the visual shortcomings of multimodal llms, 2024.

Junke Wang, Dongdong Chen, Chong Luo, Xiyang Dai, Lu Yuan, Zuxuan Wu, and Yu-Gang Jiang. Chatvideo: A tracklet-centric multimodal and versatile video understanding system, 2023a.

Yi Wang, Kunchang Li, Xinhao Li, Jiashuo Yu, Yinan He, Guo Chen, Baoqi Pei, Rongkun Zheng, Jilan Xu, Zun Wang, Yansong Shi, Tianxiang Jiang, Songze Li, Hongjie Zhang, Yifei Huang, Yu Qiao, Yali Wang, and Limin Wang. Internvideo2: Scaling video foundation models for multimodal video understanding. *ArXiv*, abs/2403.15377, 2024. URL `https://api.semanticscholar.org/CorpusID:268667436`.

Zhanyu Wang, Longyue Wang, Zhen Zhao, Minghao Wu, Chenyang Lyu, Huayang Li, Deng Cai, Luping Zhou, Shuming Shi, and Zhaopeng Tu. Gpt4video: A unified multimodal large language model for lnstruction-followed understanding and safety-aware generation, 2023b.

Le Xue, Manli Shu, Anas Awadalla, Jun Wang, An Yan, Senthil Purushwalkam, Honglu Zhou, Viraj Prabhu, Yutong Dai, Michael S Ryoo, Shrikant B. Kendre, Jieyu Zhang, Can Qin, Shu Zhen Zhang, Chia-Chih Chen, Ning Yu, Juntao Tan, Tulika Awalgaonkar, Shelby Heinecke, Huan Wang, Yejin Choi, Ludwig Schmidt, Zeyuan Chen, Silvio Savarese, Juan Carlos Niebles, Caiming Xiong, and Ran Xu. xgen-mm (blip-3): A family of open large multimodal models. *ArXiv*, abs/2408.08872, 2024. URL `https://api.semanticscholar.org/CorpusID:271891872`.

Lihe Yang, Bingyi Kang, Zilong Huang, Zhen Zhao, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything v2. *arXiv:2406.09414*, 2024.

Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. Mm-react: Prompting chatgpt for multimodal reasoning and action, 2023. URL `https://arxiv.org/abs/2303.11381`.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. Webshop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757, 2022.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL `https://arxiv.org/abs/2210.03629`.

Weihao Yu, Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Zicheng Liu, Xinchao Wang, and Lijuan Wang. Mm-vet: Evaluating large multimodal models for integrated capabilities. In *International conference on machine learning*. PMLR, 2024.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, Cong Wei, Botao Yu, Ruibin Yuan, Renliang Sun, Ming Yin, Boyuan Zheng, Zhenzhu Yang, Yibo Liu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of CVPR*, 2024.

Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, Zhiwei Liu, Yihao Feng, Tulika Awalgaonkar, Rithesh Murthy, Eric Hu, Zeyuan Chen, Ran Xu, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese, and Caiming Xiong. xlam: A family of large action models to empower ai agent systems, 2024a. URL `https://arxiv.org/abs/2409.03215`.

Jieyu Zhang, Weikai Huang, Zixian Ma, Oscar Michel, Dong He, Tanmay Gupta, Wei-Chiu Ma, Ali Farhadi, Aniruddha Kembhavi, and Ranjay Krishna. Task me anything. *arXiv preprint arXiv:2406.11775*, 2024b.

Jieyu Zhang, Le Xue, Linxin Song, Jun Wang, Weikai Huang, Manli Shu, An Yan, Zixian Ma, Juan Carlos Niebles, Silvio Savarese, Caiming Xiong, Zeyuan Chen, Ranjay Krishna, and Ran Xu. Provision: Programmatically scaling vision-centric instruction data for multimodal language models. *Preprint*, 2024c.

Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. *arXiv preprint arXiv:2401.07339*, 2024d.

A RELATED WORK

We contexualize our work in tool use and multi-modal language models.

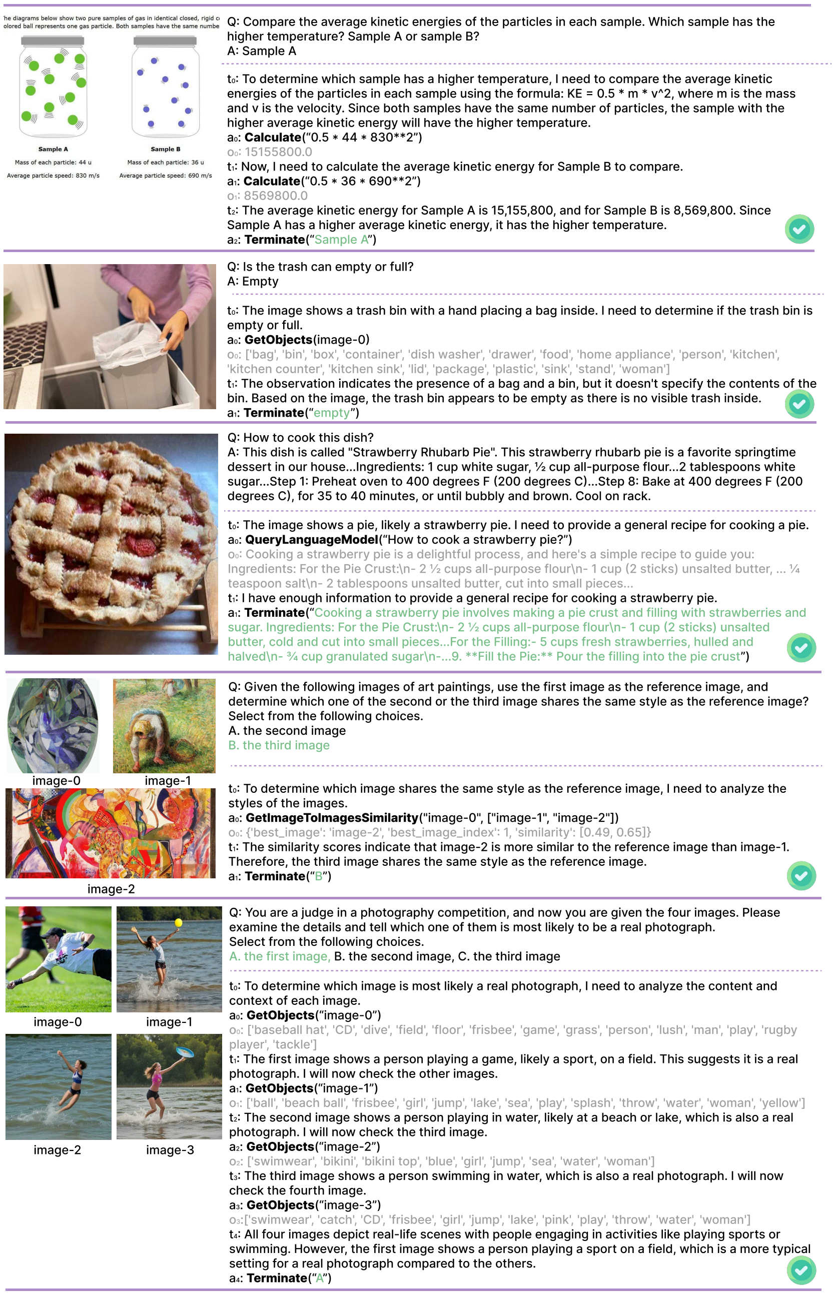
Language-only tool use. Augmenting LLMs with external tools and APIs can significantly enhance their problem-solving capabilities, especially for tasks that require specialized knowledge or interaction with external resources Qu et al. (2024). Examples include web searching Nakano et al. (2021); Yao et al. (2022), mathematical calculations He-Yueya et al. (2023); Cobbe et al. (2021), code interpretation Gao et al. (2023); Zhang et al. (2024d), and other domain-specific operations that are challenging for LLMs to accomplish solely with their intrinsic knowledge.

Researchers have mainly explored two approaches to enable LLMs to effectively leverage external tools. The first approach is through in-context learning, where instructions on tool usage and contextually relevant examples are provided directly in the prompt Gou et al. (2023); Shen et al. (2024); Yao et al. (2023); Chen et al. (2023f); Lu et al. (2024b); Paranjape et al. (2023). The second approach involves training models to specialize in calling external functions Schick et al. (2023); Qin et al. (2023); Patil et al. (2023); Chen et al. (2023a); Zhang et al. (2024a); Tang et al. (2023a). For instance, Toolformer Schick et al. (2023) proposed training LLMs to call APIs precisely to solve complex tasks in a self-supervised manner. More recently, xLAM unified data formats across various environments, and achieved more powerful LLMs specialized in function calling that are also referred to as large *action* models Zhang et al. (2024a). These works demonstrated the effectiveness of fine-tuning LLMs to execel at calling tools, which inspired our work on tuning multi-modal language models to be multi-modal action models.

Multi-modal tool-use. Unlike language-only tool use, most works in the multi-modal space have only explored training-free methods, which adapt existing LLMs to use multi-modal tools via zero-shot/few-shot prompting Gupta & Kembhavi (2022); Surís et al. (2023); Yang et al. (2023). Visprog first demonstrates the potential of leveraging LLMs for visual question answering by prompting them to generate programs of predefined modules for image understanding and manipulation Gupta & Kembhavi (2022). Simlarly, ViperGPT defines a comprehensive list of APIs and prompts code LLMs to generate Python programs for solving visual queries Surís et al. (2023). In addition to question answering, m&ms further investigated LLMs' abilities in planning and executing open-ended multi-modal tasks with external tools Ma et al. (2024). Beyond LLMs, more recent efforts such as VisualSketchPad also augmented multi-modal models with external tools and showcased success Hu et al. (2024a). However, in-context tool use only works well with powerful proprietary multi-modal models such as GPT-4o; all open-source multi-modal models fail to perform tool use well without fine-tuning.

There are only a few works on training multi-modal models to be better at tool use. LLaVa-Plus was the first work that shows the possibility of training a multi-modal model to use vision specialists Liu et al. (2023c). However, it exhibits poor performance due to the weakness of the model backbone and the small scale and suboptimal quality of the data Liu et al. (2023c). Visual Program Distillation distills tool-use and reasoning abilities into a multi-modal model with chain-of-thought (CoT) data obtained from programs Hu et al. (2023). Nonetheless, this model is unable to use tools at inference and is limited to simple question answering tasks Hu et al. (2023). Similarly, the more recent work Visual CoT introduces a new synthetic CoT dataset for training multi-modal models for enhanced reasoning Shao et al. (2024). The closest work to ours is CogCoM, which identifies 6 useful manipulations and trains multi-modal models with synthetic chain-of-manipulation (CoM) data Qi et al. (2024). Nonetheless, the manipulations are limited and useful for only detailed visual question answering and visual grounding, and the authors have only experimented with adding 70K CoM data to 500K QA examples. None of these works studied or improved the quantity and quality of these chains to enhance multi-modal models' performance across diverse complex multi-modal tasks, which our work focuses on.

Multi-modal language models. Most recently, there have been many advances made on open-source multi-modal models Wang et al. (2024); Huang et al. (2023); Chen et al. (2023b); Wang et al. (2023b); Sun et al. (2023); Lyu et al. (2023); Tang et al. (2023b); Wang et al. (2023a); Lin et al. (2023); Bi et al. (2023); Chen et al. (2023c); Liu et al. (2024b); Peng et al. (2023); Chen et al. (2023d); Shukor et al. (2023); Lin et al. (2023); Lu et al. (2023); Li et al. (2023a); Sun et al. (2024b); Moor et al. (2023); Awadalla et al. (2023); Sun et al. (2024a); Chen et al. (2023e; 2024b); Liu et al. (2023b;a; 2024a); Dai et al. (2024); Li et al. (2022; 2023c); Panagopoulou et al. (2023); Xue et al.


Sample A
Mass of each particle: 44 u
Average particle speed: 830 m/s
Sample B
Mass of each particle: 36 u
Average particle speed: 690 m/s
image-0
image-1
image-2
image-0
image-1
image-2
image-3

Q: Compare the average kinetic energies of the particles in each sample. Which sample has the higher temperature? Sample A or sample B?
A: Sample A

t_0: To determine which sample has a higher temperature, I need to compare the average kinetic energies of the particles in each sample using the formula: KE = 0.5 * m * v^2, where m is the mass and v is the velocity. Since both samples have the same number of particles, the sample with the higher average kinetic energy will have the higher temperature.
a_0: **Calculate**("0.5 * 44 * 830**2")
o_0: 15155800.0
t_1: Now, I need to calculate the average kinetic energy for Sample B to compare.
a_1: **Calculate**("0.5 * 36 * 690**2")
o_1: 8569800.0
t_2: The average kinetic energy for Sample A is 15,155,800, and for Sample B is 8,569,800. Since Sample A has a higher average kinetic energy, it has the higher temperature.
a_2: **Terminate**("Sample A")

Q: Is the trash can empty or full?
A: Empty

t_0: The image shows a trash bin with a hand placing a bag inside. I need to determine if the trash bin is empty or full.
a_0: **GetObjects**(image-0)
o_0: ['bag', 'bin', 'box', 'container', 'dish washer', 'drawer', 'food', 'home appliance', 'person', 'kitchen', 'kitchen counter', 'kitchen sink', 'lid', 'package', 'plastic', 'sink', 'stand', 'woman']
t_1: The observation indicates the presence of a bag and a bin, but it doesn't specify the contents of the bin. Based on the image, the trash bin appears to be empty as there is no visible trash inside.
a_1: **Terminate**("empty")

Q: How to cook this dish?
A: This dish is called "Strawberry Rhubarb Pie". This strawberry rhubarb pie is a favorite springtime dessert in our house...Ingredients: 1 cup white sugar, ½ cup all-purpose flour...2 tablespoons white sugar...Step 1: Preheat oven to 400 degrees F (200 degrees C)...Step 8: Bake at 400 degrees F (200 degrees C), for 35 to 40 minutes, or until bubbly and brown. Cool on rack.

t_0: The image shows a pie, likely a strawberry pie. I need to provide a general recipe for cooking a pie.
a_0: **QueryLanguageModel**("How to cook a strawberry pie?")
o_0: Cooking a strawberry pie is a delightful process, and here's a simple recipe to guide you: Ingredients: For the Pie Crust:\n- 2 ½ cups all-purpose flour\n- 1 cup (2 sticks) unsalted butter, ... ¼ teaspoon salt\n- 2 tablespoons unsalted butter, cut into small pieces...
t_1: I have enough information to provide a general recipe for cooking a strawberry pie.
a_1: **Terminate**("Cooking a strawberry pie involves making a pie crust and filling with strawberries and sugar. Ingredients: For the Pie Crust:\n- 2 ½ cups all-purpose flour\n- 1 cup (2 sticks) unsalted butter, cold and cut into small pieces...For the Filling:- 5 cups fresh strawberries, hulled and halved\n- ¾ cup granulated sugar\n-...9. **Fill the Pie:** Pour the filling into the pie crust")

Q: Given the following images of art paintings, use the first image as the reference image, and determine which one of the second or the third image shares the same style as the reference image? Select from the following choices.
A. the second image
B. the third image

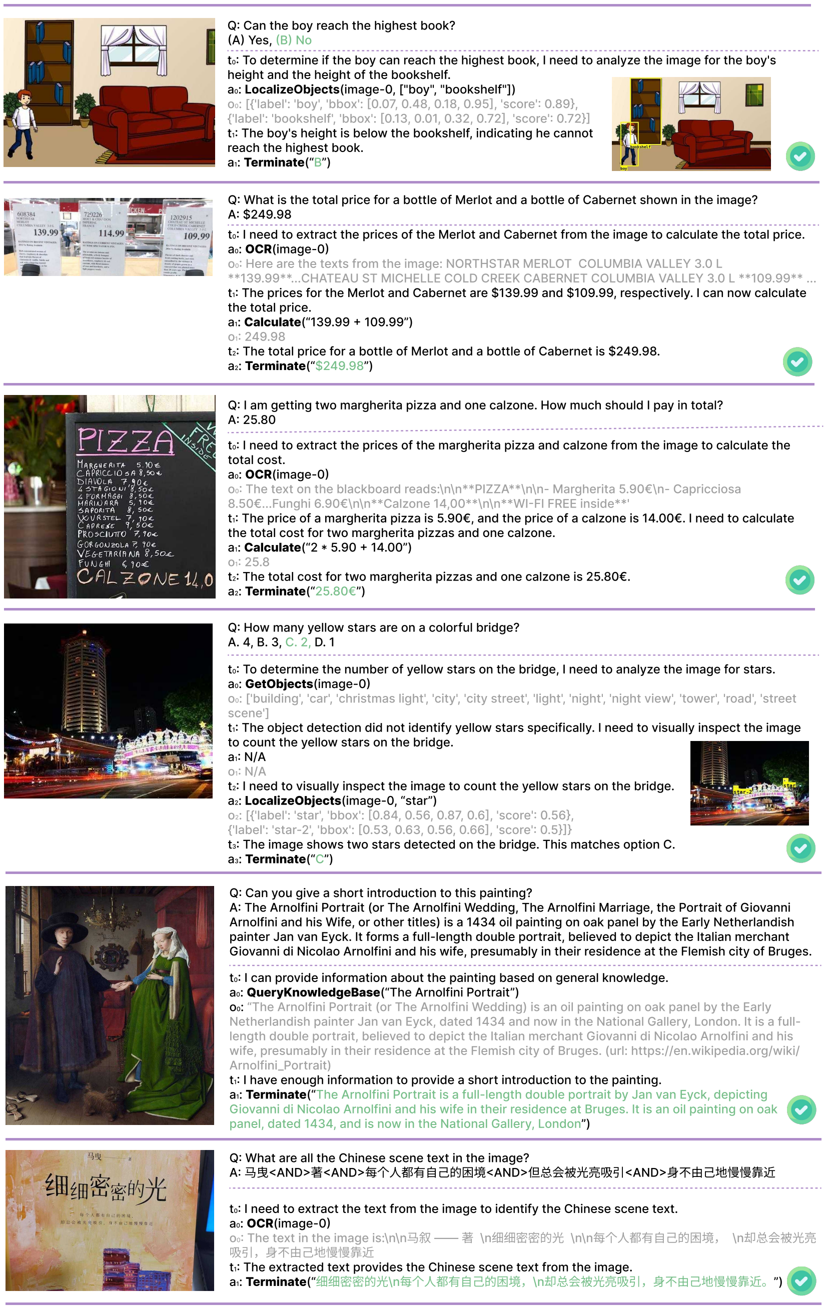
t_0: To determine which image shares the same style as the reference image, I need to analyze the styles of the images.
a_0: **GetImageToImagesSimilarity**("image-0", ["image-1", "image-2"])
o_0: {'best_image': 'image-2', 'best_image_index': 1, 'similarity': [0.49, 0.65]}
t_1: The similarity scores indicate that image-2 is more similar to the reference image than image-1. Therefore, the third image shares the same style as the reference image.
a_1: **Terminate**("B")

Q: You are a judge in a photography competition, and now you are given the four images. Please examine the details and tell which one of them is most likely to be a real photograph.
Select from the following choices.
A. the first image, B. the second image, C. the third image

t_0: To determine which image is most likely a real photograph, I need to analyze the content and context of each image.
a_0: **GetObjects**("image-0")
o_0: ['baseball hat', 'CD', 'dive', 'field', 'floor', 'frisbee', 'game', 'grass', 'person', 'lush', 'man', 'play', 'rugby player', 'tackle']
t_1: The first image shows a person playing a game, likely a sport, on a field. This suggests it is a real photograph. I will now check the other images.
a_1: **GetObjects**("image-1")
o_1: ['ball', 'beach ball', 'frisbee', 'girl', 'jump', 'lake', 'sea', 'play', 'splash', 'throw', 'water', 'woman', 'yellow']
t_2: The second image shows a person playing in water, likely at a beach or lake, which is also a real photograph. I will now check the third image.
a_2: **GetObjects**("image-2")
o_2: ['swimwear', 'bikini', 'bikini top', 'blue', 'girl', 'jump', 'sea', 'water', 'woman']
t_3: The third image shows a person swimming in water, which is also a real photograph. I will now check the fourth image.
a_3: **GetObjects**("image-3")
o_3:['swimwear', 'catch', 'CD', 'frisbee', 'girl', 'jump', 'lake', 'pink', 'play', 'throw', 'water', 'woman']
t_4: All four images depict real-life scenes with people engaging in activities like playing sports or swimming. However, the first image shows a person playing a sport on a field, which is a more typical setting for a real photograph compared to the others.
a_4: **Terminate**("A")

Figure 4: We present additional successful examples of TACO.

(2024). These efforts include training multi-modal models to take in multiple images, engage in multi-turn conversations, and even understand videos Liu et al. (2024a); Jiang et al. (2024); Li et al. (2024). For example, LLaVA-Next achieves strong multi-image understanding through large-scale interleaved visual instruction tuning with M4-Instruct Liu et al. (2024a). Similarly, Mantis introduces a new large-scale multi-image instruction tuning dataset Mantis-Instruct for multi-image training Jiang et al. (2024). These efforts pave the foundation for our work on training multi-modal models to learn from multi-image interleaved chains-of-thought-and-action data.



Q: Can the boy reach the highest book?
(A) Yes, (B) No

t_0: To determine if the boy can reach the highest book, I need to analyze the image for the boy's height and the height of the bookshelf.
a_0: **LocalizeObjects**(image-0, ["boy", "bookshelf"])
o_0: [{'label': 'boy', 'bbox': [0.07, 0.48, 0.18, 0.95], 'score': 0.89}, {'label': 'bookshelf', 'bbox': [0.13, 0.01, 0.32, 0.72], 'score': 0.72}]
t_1: The boy's height is below the bookshelf, indicating he cannot reach the highest book.
a_1: **Terminate**("B")

Q: What is the total price for a bottle of Merlot and a bottle of Cabernet shown in the image?
A: $249.98

t_0: I need to extract the prices of the Merlot and Cabernet from the image to calculate the total price.
a_0: **OCR**(image-0)
o_0: Here are the texts from the image: NORTHSTAR MERLOT COLUMBIA VALLEY 3.0 L **139.99**...CHATEAU ST MICHELLE COLD CREEK CABERNET COLUMBIA VALLEY 3.0 L **109.99** ...
t_1: The prices for the Merlot and Cabernet are $139.99 and $109.99, respectively. I can now calculate the total price.
a_1: **Calculate**("139.99 + 109.99")
o_1: 249.98
t_2: The total price for a bottle of Merlot and a bottle of Cabernet is $249.98.
a_2: **Terminate**("$249.98")

Q: I am getting two margherita pizza and one calzone. How much should I pay in total?
A: 25.80

t_0: I need to extract the prices of the margherita pizza and calzone from the image to calculate the total cost.
a_0: **OCR**(image-0)
o_0: The text on the blackboard reads:\n\n**PIZZA**\n\n- Margherita 5.90€\n- Capricciosa 8.50€...Funghi 6.90€\n\n**Calzone 14,00**\n\n**WI-FI FREE inside**'
t_1: The price of a margherita pizza is 5.90€, and the price of a calzone is 14.00€. I need to calculate the total cost for two margherita pizzas and one calzone.
a_1: **Calculate**("2 ∗ 5.90 + 14.00")
o_1: 25.8
t_2: The total cost for two margherita pizzas and one calzone is 25.80€.
a_2: **Terminate**("25.80€")

Q: How many yellow stars are on a colorful bridge?
A. 4, B. 3, C. 2, D. 1

t_0: To determine the number of yellow stars on the bridge, I need to analyze the image for stars.
a_0: **GetObjects**(image-0)
o_0: ['building', 'car', 'christmas light', 'city', 'city street', 'light', 'night', 'night view', 'tower', 'road', 'street scene']
t_1: The object detection did not identify yellow stars specifically. I need to visually inspect the image to count the yellow stars on the bridge.
a_1: N/A
o_1: N/A
t_2: I need to visually inspect the image to count the yellow stars on the bridge.
a_2: **LocalizeObjects**(image-0, "star")
o_2: [{'label': 'star', 'bbox': [0.84, 0.56, 0.87, 0.6], 'score': 0.56}, {'label': 'star-2', 'bbox': [0.53, 0.63, 0.56, 0.66], 'score': 0.5}]}
t_3: The image shows two stars detected on the bridge. This matches option C.
a_3: **Terminate**("C")

Q: Can you give a short introduction to this painting?
A: The Arnolfini Portrait (or The Arnolfini Wedding, The Arnolfini Marriage, the Portrait of Giovanni Arnolfini and his Wife, or other titles) is a 1434 oil painting on oak panel by the Early Netherlandish painter Jan van Eyck. It forms a full-length double portrait, believed to depict the Italian merchant Giovanni di Nicolao Arnolfini and his wife, presumably in their residence at the Flemish city of Bruges.

t_0: I can provide information about the painting based on general knowledge.
a_0: **QueryKnowledgeBase**("The Arnolfini Portrait")
o_0: "The Arnolfini Portrait (or The Arnolfini Wedding) is an oil painting on oak panel by the Early Netherlandish painter Jan van Eyck, dated 1434 and now in the National Gallery, London. It is a full-length double portrait, believed to depict the Italian merchant Giovanni di Nicolao Arnolfini and his wife, presumably in their residence at the Flemish city of Bruges. (url: https://en.wikipedia.org/wiki/Arnolfini_Portrait)
t_1: I have enough information to provide a short introduction to the painting.
a_1: **Terminate**("The Arnolfini Portrait is a full-length double portrait by Jan van Eyck, depicting Giovanni di Nicolao Arnolfini and his wife in their residence at Bruges. It is an oil painting on oak panel, dated 1434, and is now in the National Gallery, London")

Q: What are all the Chinese scene text in the image?
A: 马曳<AND>著<AND>每个人都有自己的困境<AND>但总会被光亮吸引<AND>身不由己地慢慢靠近

t_0: I need to extract the text from the image to identify the Chinese scene text.
a_0: **OCR**(image-0)
o_0: The text in the image is:\n\n马叙 —— 著 \n细细密密的光 \n\n每个人都有自己的困境， \n却总会被光亮吸引，身不由己地慢慢靠近
t_1: The extracted text provides the Chinese scene text from the image.
a_1: **Terminate**("细细密密的光\n每个人都有自己的困境，\n却总会被光亮吸引，身不由己地慢慢靠近。")

Figure 5: We present additional successful examples of TACO.

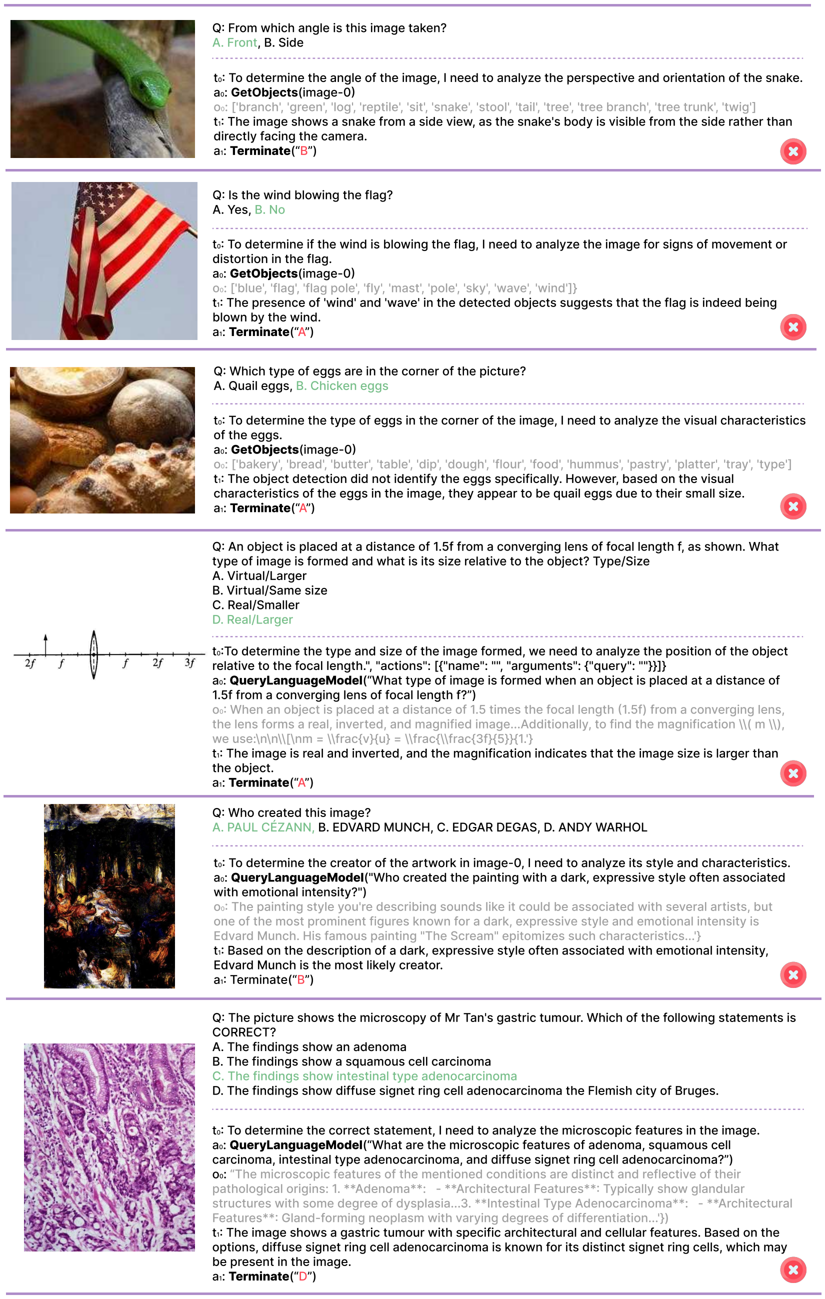
B ADDITIONAL RESULTS

B.1 QUALITATIVE EXAMPLES

We present additional successful outputs of TACO across both single-image and multi-image examples in Figures 4 and 5 as well as failure cases in Figure 6.

B.2 QUANTITATIVE RESULTS

Quality matters more than quantity: our best data recipe is the smallest and yet leads to better performance than other recipes of larger scales. We explore two data filtering techniques to adjust the distribution of (1) data formats and (2) data sources (Table 3). We find that 293K CoTA



Q: From which angle is this image taken?
A. Front, B. Side

t_0: To determine the angle of the image, I need to analyze the perspective and orientation of the snake.
a_0: **GetObjects**(image-0)
o_0: ['branch', 'green', 'log', 'reptile', 'sit', 'snake', 'stool', 'tail', 'tree', 'tree branch', 'tree trunk', 'twig']
t_1: The image shows a snake from a side view, as the snake's body is visible from the side rather than directly facing the camera.
a_1: **Terminate**("B")

Q: Is the wind blowing the flag?
A. Yes, B. No

t_0: To determine if the wind is blowing the flag, I need to analyze the image for signs of movement or distortion in the flag.
a_0: **GetObjects**(image-0)
o_0: ['blue', 'flag', 'flag pole', 'fly', 'mast', 'pole', 'sky', 'wave', 'wind']}
t_1: The presence of 'wind' and 'wave' in the detected objects suggests that the flag is indeed being blown by the wind.
a_1: **Terminate**("A")

Q: Which type of eggs are in the corner of the picture?
A. Quail eggs, B. Chicken eggs

t_0: To determine the type of eggs in the corner of the image, I need to analyze the visual characteristics of the eggs.
a_0: **GetObjects**(image-0)
o_0: ['bakery', 'bread', 'butter', 'table', 'dip', 'dough', 'flour', 'food', 'hummus', 'pastry', 'platter', 'tray', 'type']
t_1: The object detection did not identify the eggs specifically. However, based on the visual characteristics of the eggs in the image, they appear to be quail eggs due to their small size.
a_1: **Terminate**("A")

Q: An object is placed at a distance of 1.5f from a converging lens of focal length f, as shown. What type of image is formed and what is its size relative to the object? Type/Size
A. Virtual/Larger
B. Virtual/Same size
C. Real/Smaller
D. Real/Larger

t_0:To determine the type and size of the image formed, we need to analyze the position of the object relative to the focal length.", "actions": [{"name": "", "arguments": {"query": ""}}]}
a_0: **QueryLanguageModel**("What type of image is formed when an object is placed at a distance of 1.5f from a converging lens of focal length f?")
o_0: When an object is placed at a distance of 1.5 times the focal length (1.5f) from a converging lens, the lens forms a real, inverted, and magnified image...Additionally, to find the magnification \\(m \\), we use:\n\n\\[\nm = \\frac{v}{u} = \\frac{\\frac{3f}{5}}{1.'}
t_1: The image is real and inverted, and the magnification indicates that the image size is larger than the object.
a_1: **Terminate**("A")

Q: Who created this image?
A. PAUL CÉZANN, B. EDVARD MUNCH, C. EDGAR DEGAS, D. ANDY WARHOL

t_0: To determine the creator of the artwork in image-0, I need to analyze its style and characteristics.
a_0: **QueryLanguageModel**("Who created the painting with a dark, expressive style often associated with emotional intensity?")
o_0: The painting style you're describing sounds like it could be associated with several artists, but one of the most prominent figures known for a dark, expressive style and emotional intensity is Edvard Munch. His famous painting "The Scream" epitomizes such characteristics...'}
t_1: Based on the description of a dark, expressive style often associated with emotional intensity, Edvard Munch is the most likely creator.
a_1: Terminate("B")

Q: The picture shows the microscopy of Mr Tan's gastric tumour. Which of the following statements is CORRECT?
A. The findings show an adenoma
B. The findings show a squamous cell carcinoma
C. The findings show intestinal type adenocarcinoma
D. The findings show diffuse signet ring cell adenocarcinoma the Flemish city of Bruges.

t_0: To determine the correct statement, I need to analyze the microscopic features in the image.
a_0: **QueryLanguageModel**("What are the microscopic features of adenoma, squamous cell carcinoma, intestinal type adenocarcinoma, and diffuse signet ring cell adenocarcinoma?")
o_0: "The microscopic features of the mentioned conditions are distinct and reflective of their pathological origins: 1. **Adenoma**:   - **Architectural Features**: Typically show glandular structures with some degree of dysplasia...3. **Intestinal Type Adenocarcinoma**:   - **Architectural Features**: Gland-forming neoplasm with varying degrees of differentiation...'})
t_1: The image shows a gastric tumour with specific architectural and cellular features. Based on the options, diffuse signet ring cell adenocarcinoma is known for its distinct signet ring cells, which may be present in the image.
a_1: **Terminate**("D")

Figure 6: We present additional failure cases of TACO.

examples result in the best absolute performance and the biggest gains of 3.6% over the baseline (Table 3). Adding CoT examples results in a smaller gain of 1.7%, even though the training data size almost doubles (Table 3). On the other hand, combining CoTA and Direct examples hurts the model's performance, likely because mixing in the Direct examples weakens the model's reasoning and and action calling abilities compared to CoTA only data.

Data sources matter too: including all datasets hurts performance whereas including only Action-useful datasets brings performance gains. Similarly, we see that including only the Action-useful datasets – where GPT-4o frequently chooses to call actions and reach correct final answers – improves the model's average performance compared to the baseline, while including all data sources does not (Table 3). Again, we see that a smaller set of 566K CoTA traces can lead to

Table 3: **Model-generated data ablations.** Data quality matters more than quantity. We find that (1) the smallest dataset with only CoTA examples results in better average performance and higher gains compared to other larger datasets with a mix of CoTA and Direct examples; (2) filtering out Action-useless datasets leads to performance gains.

Data source	Final data format	Size	Model	A-OKVQA	BLINK	MathVista	MMMU	MMStar	MMVet	MMVP	RealWorldQA	Avg	Delta
All datasets	Direct	293K	Mantis-SigLIP	80.3	43.7	31.1	40.4	40.5	33.0	63.3	51.8	48.0	
	CoTA		TACO	82.4	**47.8**	34.9	40.3	**44.6**	**45.2**	64.0	53.7	**51.6**	**+3.6**
	Direct	580K	Mantis-SigLIP	82.3	45.2	34.2	**42.6**	39.5	31.9	**67.7**	52.6	49.5	
	CoTA+ CoT		TACO	**84.0**	46.4	**36.3**	40.3	40.6	43.7	66.7	51.6	51.2	+1.7
	Direct	528K	Mantis-SigLIP	79.7	46.7	34.0	39.4	40.7	28.3	65.0	52.2	48.2	
	CoTA+ Direct		TACO	79.1	45.3	34.0	40.1	38.0	33.3	61.0	50.5	47.7	-0.6
	Direct	815K	Mantis-SigLIP	81.0	46.9	35.0	39.9	39.7	29.5	66.7	**54.5**	49.1	
	CoTA+ CoT+ Direct		TACO	81.4	45.2	33.8	39.7	38.3	33.1	64.7	50.1	48.3	-0.9
Action-useful datasets	Direct	566K	Mantis-SigLIP	81.6	42.4	32.8	42.2	40.3	26.8	67.0	50.2	47.9	
	CoTA+ CoT+ Direct		TACO	82.4	43.3	31.9	38.1	39.5	35.6	67.0	50.3	48.5	+0.6

Table 4: **Model-generated and program-generated data mixtures.** Adding programmatically generated CoTA data can increase the model's performance on some benchmarks such as A-OKVQA, MathVista, MMMU, and MMVP. However, it doesn't further improve model's average performance across all benchmarks. Additionally, more programmatic CoTA can even hurt the model's performance.

M:P = Model-generated CoTA (M-CoTA): Program-generated CoTA (P-CoTA).

Model		Train data	Total size	A-OKVQA	BLINK	MathVista	MMMU	MMStar	MMVet	MMVP	RealWorldQA	Avg	Delta
Mantis-SigLIP		Direct 293K	293K	80.3	43.7	31.1	40.4	40.5	33.0	63.3	51.8	48.0	
TACO	M:P	M-CoTA / P-CoTA											
	0:1	0 / 293K	293K	34.3	37.4	17.3	31.9	30.4	0.0	48.3	40.7	30.0	—
	1:0	293 / 0K		82.4	**47.8**	34.9	40.3	**44.6**	**45.2**	64.0	**53.7**	**51.6**	**+3.6**
	1:0.1	293 / 29K	322K	**82.6**	47.5	33.9	40.3	44.2	42.3	64.3	49.8	50.6	+2.6
	1:0.25	293 / 73K	366K	82.1	44.2	**38.3**	40.2	42.9	45.1	**64.7**	51.2	51.1	+3.1
	1:0.5	293 / 147K	440K	81.9	46.0	36.7	**41.4**	41.4	40.9	62.3	50.3	50.1	+2.1
	1:1	293 / 293K	586K	81.1	47.7	31.0	39.3	41.4	36.2	63.0	50.7	48.8	+0.8

better performance than a much larger dataset (815K), implying that data quality matters more than quantity.

Adding programmatically generated data can lead to further gains on some benchmarks but brings no additional gains to the average performance. We have also trained TACO with different mixtures of model-generated and program-generated data, where we vary their relative ratios from 1:0.1 to 1:1. We learn that additional programmatic data can bring gains on some benchmarks such as A-OKVQA, MathVista, MMMU and MMVP (Table 4). However, none of the data mixtures results in better average performance than the best model-generated data. We present a qualitative analysis of these findings in Section B.3.

Additional gains can be achieved by tuning the vision encoder, training with a smaller learning rate or for more epochs. Last but not least, our hyperparameter tuning experiments with LLaVa-OV-Stage1.5 suggest that we can further improve the model's absolute performance by tuning the vision encoder, training with a smaller learning rate and/or for longer epochs (Figure 7).

Table 5: **Additional Results on Model-generated data ablations with Mantis-CLIP.** We observe similar results of data ablations on Mantis-CLIP as on Mantis-SigLIP.

Data source	Final data format	Size	Model	A-OKVQA	BLINK	MathVista	MMMU	MMStar	MMVet	MMVP	RealWorldQA	Avg	Delta
All datasets	Direct	293K	Mantis-CLIP	80.7	45.8	33.1	42.2	36.7	28.9	62.7	**52.3**	47.8	
	CoTA		TACO	81.1	**49.6**	**36.6**	42.8	**40.8**	**45.2**	**63.3**	51.1	**51.3**	**3.5**
	Direct	580K	Mantis-CLIP	82.0	47.2	31.5	40.6	38.3	31.4	**63.3**	49.0	47.9	
	CoTA+ CoT		TACO	**82.6**	47.7	35.9	38.3	39.6	43.4	61.0	51.1	49.9	2.0
	Direct	815K	Mantis-CLIP	81.0	47.3	32.9	47.3	38.1	29.1	63.0	51.0	48.7	
	CoTA+ CoT+ Direct		TACO	81.3	43.1	32.7	**48.0**	37.7	31.0	58.7	47.6	47.5	-1.2
Action-useful datasets	Direct	566K	Mantis-CLIP	79.1	44.1	30.9	39.4	35.5	30.1	62.7	49.8	46.5	
	CoTA+ CoT+ Direct		TACO	80.7	41.6	31.5	41.0	38.2	36.1	59.0	48.2	47.0	0.6

We report additional quantitative results of data ablatins on Mantis-CLIP in Table 5, where we see the same trends we observe with Mantis-SigLIP: the smallest dataset of 293K CoTA examples leads to the highest absolute performance and gain compared to other datasets with a mix of CoTA, CoT, and/or Direct examples at larger scales.

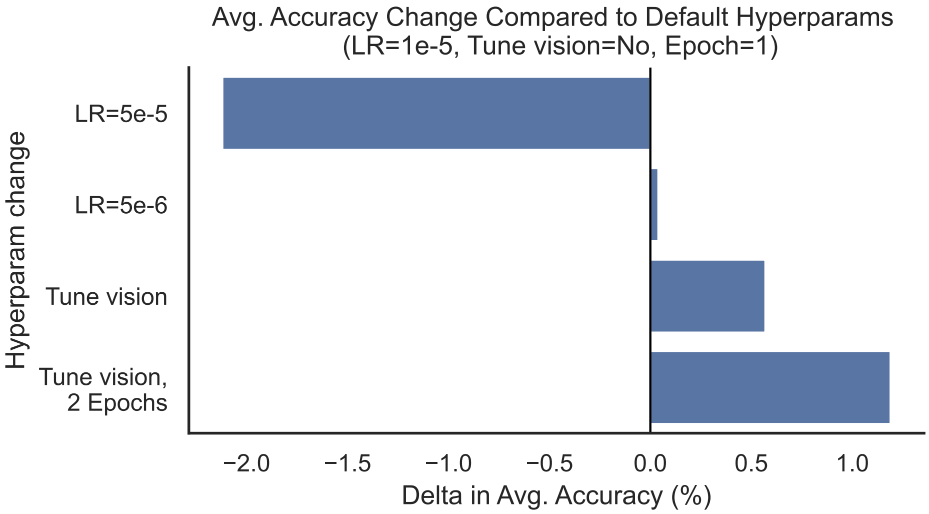
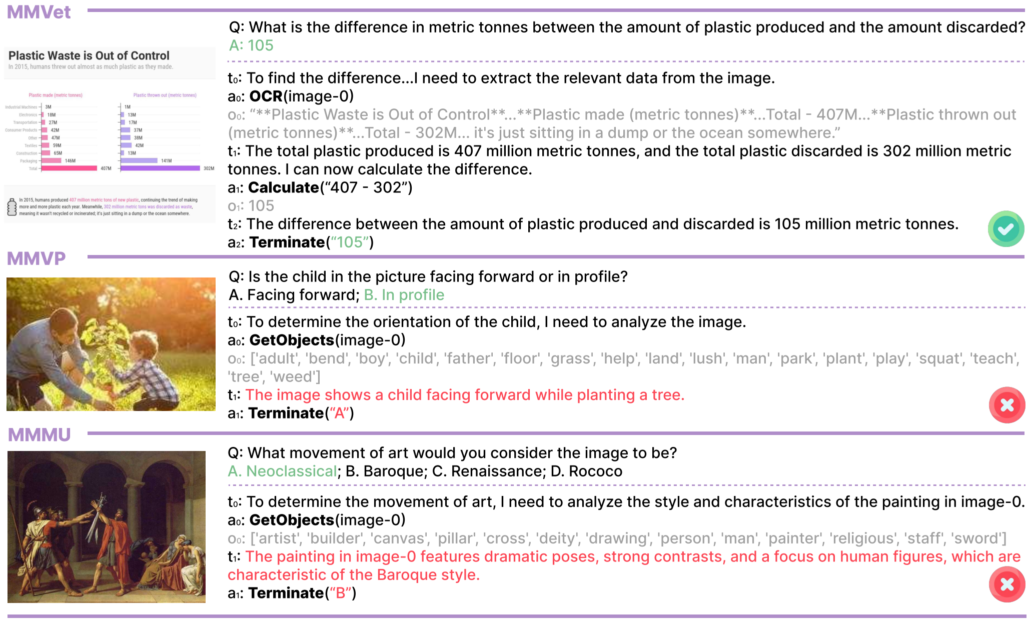


Figure 7: **Hyperparameter ablations.** Additional gains can be achieved with a smaller learning rate for the language model, tuning the vision encoder, and training for more epochs.



Q: What is the difference in metric tonnes between the amount of plastic produced and the amount discarded?
A: 105

t_0: To find the difference...I need to extract the relevant data from the image.
a_0: **OCR**(image-0)
o_0: "**Plastic Waste is Out of Control**...**Plastic made (metric tonnes)**...Total - 407M...**Plastic thrown out (metric tonnes)**...Total - 302M... it's just sitting in a dump or the ocean somewhere."
t_1: The total plastic produced is 407 million metric tonnes, and the total plastic discarded is 302 million metric tonnes. I can now calculate the difference.
a_1: **Calculate**("407 - 302")
o_1: 105
t_2: The difference between the amount of plastic produced and discarded is 105 million metric tonnes.
a_2: **Terminate**("105")

Q: Is the child in the picture facing forward or in profile?
A. Facing forward; B. In profile

t_0: To determine the orientation of the child, I need to analyze the image.
a_0: **GetObjects**(image-0)
o_0: ['adult', 'bend', 'boy', 'child', 'father', 'floor', 'grass', 'help', 'land', 'lush', 'man', 'park', 'plant', 'play', 'squat', 'teach', 'tree', 'weed']
t_1: The image shows a child facing forward while planting a tree.
a_1: **Terminate**("A")

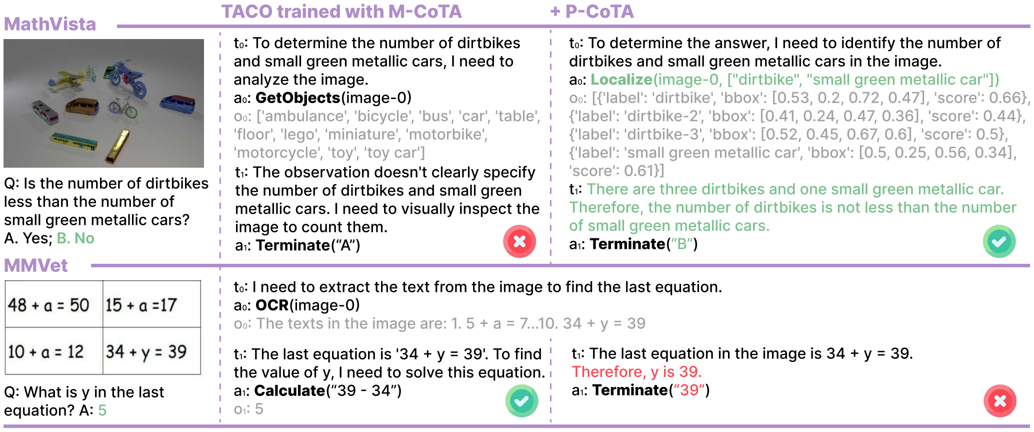
Q: What movement of art would you consider the image to be?
A. Neoclassical; B. Baroque; C. Renaissance; D. Rococo

t_0: To determine the movement of art, I need to analyze the style and characteristics of the painting in image-0.
a_0: **GetObjects**(image-0)
o_0: ['artist', 'builder', 'canvas', 'pillar', 'cross', 'deity', 'drawing', 'person', 'man', 'painter', 'religious', 'staff', 'sword']
t_1: The painting in image-0 features dramatic poses, strong contrasts, and a focus on human figures, which are characteristic of the Baroque style.
a_1: **Terminate**("B")

Figure 8: We present representative examples of TACO successes on MMVet, where our model achieves consistent large gains, and failures on MMVP and MMMU, where our data sometimes results in small drops.

B.3 QUALITATIVE ANALYSIS

Why does CoTA data help MMVet a lot but can harm performance on MMVP / MMMU? We find that CoTA brings huge gains on MMVet, which includes many questions involving OCR and math that our model is able to answer correctly much more often by calling OCR and CALCULATE (Figure 8). By contrast, we learn that CoTA sometimes decreases model's performance on MMVP and MMMU, because our action space doesn't cover tools that would be directly helpful for many questions in these datasets (e.g. orientation questions in MMVP, and domain-specific questions in MMMU) (Figure 8). In these cases, if the model called actions, the action outputs would be useless or even misleading (Figure 8).



Q: Is the number of dirtbikes less than the number of small green metallic cars? A. Yes; B. No

t_0: To determine the number of dirtbikes and small green metallic cars, I need to analyze the image.
a_0: **GetObjects**(image-0)
o_0: ['ambulance', 'bicycle', 'bus', 'car', 'table', 'floor', 'lego', 'miniature', 'motorbike', 'motorcycle', 'toy', 'toy car']
t_1: The observation doesn't clearly specify the number of dirtbikes and small green metallic cars. I need to visually inspect the image to count them.
a_1: **Terminate**("A") ✗

t_0: To determine the answer, I need to identify the number of dirtbikes and small green metallic cars in the image.
a_0: **Localize**(image-0, ["dirtbike", "small green metallic car"])
o_0: [{'label': 'dirtbike', 'bbox': [0.53, 0.2, 0.72, 0.47], 'score': 0.66}, {'label': 'dirtbike-2', 'bbox': [0.41, 0.24, 0.47, 0.36], 'score': 0.44}, {'label': 'dirtbike-3', 'bbox': [0.52, 0.45, 0.67, 0.6], 'score': 0.5}, {'label': 'small green metallic car', 'bbox': [0.5, 0.25, 0.56, 0.34], 'score': 0.61}]
t_1: There are three dirtbikes and one small green metallic car. Therefore, the number of dirtbikes is not less than the number of small green metallic cars.
a_1: **Terminate**("B") ✓

Q: What is y in the last equation? A: 5

t_0: I need to extract the text from the image to find the last equation.
a_0: **OCR**(image-0)
o_0: The texts in the image are: 1. 5 + a = 7...10. 34 + y = 39

t_1: The last equation is '34 + y = 39'. To find the value of y, I need to solve this equation.
a_1: **Calculate**("39 - 34")
o_1: 5 ✓

t_1: The last equation in the image is 34 + y = 39. Therefore, y is 39.
a_1: **Terminate**("39") ✗

Figure 9: We present examples of TACO success and failure after adding programmatic data to the fine-tuning dataset.

Why does adding programmatic CoTA help on MathVista but hurt MMVet performance? We observe that adding programmatic CoTA can result in up to 3% gain on MathVista and 9% drop on MMVet (Table 4). Upon analysis, we discover that programmatic CoTA improves the general VQA split in MathVista the most by almost 9%. This is because LOCALIZE is helpful for these questions, and our programmatic data includes many LOCALIZE instances that allow TACO to learn to use it effectively (Figure 9). Conversely, programmatic data hurts TACO's performance on MMVet most likely due to the model's worse reasoning ability as a result of the simple and rigid thoughts generated with templates in our programmatic data (Figure 9). See the Appendix for more qualitative examples.

Taken together, these findings suggest that the quality of the thoughts and diversity of actions are both crucial to the model's performance.

C ADDITIONAL METHOD DETAILS

C.1 ACTION SPACE

We define the action space A of our multi-modal action model as the set of atomic tools useful for multi-step multi-modal tasks. We choose the tools from prior work on multi-modal tool use Hu et al. (2024a); Surís et al. (2023); Gupta & Kembhavi (2022); Liu et al. (2023c); Ma et al. (2024), combining all commonly used tools, standardizing their names and adapting them to work with flexible input and output formats. Our final set of 15 tools includes OCR, GETOBJECTS, LOCALIZEOBJECTS, ESTIMATEOBJECTDEPTH, ESTIMATEREGIONDEPTH, GETIMAGETOTEXTSSIMILARITY, GETIMAGETOIMAGESSIMILARITY, GETTEXTTOIMAGESSIMILARITY, DETECTFACES, CROP, ZOOMIN, QUERYLANGUAGEMODEL, QUERYKNOWLEDGEBASE, CALCULATE, and SOLVEMATHEQUATION. Note that most tools in this list are vision-centric or visual-linguistic tools; the last four tools are language and math tools useful for general reasoning or information retrieval, which are also often involved in multi-modal tasks. We also include the TERMINATE tool for the model to output a final answer in the same action format. See implementation details in the Appendix.

C.2 ACTION IMPLEMENTATION

Our Python implementation of all actions can be found in Listing 1.

C.3 DATA FILTERING AND MIXING.

In total, we generate 815K data with GPT-4o for both single-image and multi-image questions across 31 data sources from Cauldron and Mantis-Instruct Laurençon et al. (2024); Jiang et al. (2024). We also programmatically generate 1M+ QA pairs and corresponding CoTA data with VG images and annotations Krishna et al. (2016), which we randomly sample from to augment model-generated data. We then develop 9 different data recipes with 3 filtering/mixing techniques, where we vary the distribution of (1) data formats, (2) data sources, and (3) model-generated vs. program-generated data.

Data format. As mentioned in the data generation pipeline, model-generated data can be categorized into two formats: Chain-of-Thought-and-Action (CoTA) or Chain-of-Thought (CoT) examples (Figure 2). Additionally, they can be further grouped into CoTA/CoT-pos and CoTA/CoT-neg examples where the final answers are correct and wrong respectively (Figure 3). Note that we convert both CoTA-neg and CoT-neg examples into the direct answer (Direct) format with groundtruth answers (Figure 2) so the final data format is one of CoTA, CoT, and Direct. We experiment with data consisting of only CoTA, CoTA and CoT, CoTA and Direct, and all three formats (Figure 3).

Data source. We also perform filtering based on data sources as Cauldron and Mantis-Instruct cover a wide range of tasks, some of which benefit more from our defined actions than others. To this end, we define Action-useless datasets as the ones where GPT-4o either decides to output only thoughts much more often than both thoughts and actions (i.e. % of CoT-pos $-$ CoTA-pos > 10), or reaches wrong answers much more frequently than correct ones when calling actions (i.e. % CoTA-neg $-$ CoTA-pos > 10) (Figure 3). The remaining datasets are considered Action-useful datasets, and we experiment with including all datasets vs. only the Action-useful datasets (Figure 3).

Model-generated vs. program-generated data. Due to the lack of controllability, the distribution of actions in model-generated data is highly imbalanced, with a couple of actions such as GETOBJECTS and OCR dominating the dataset. Therefore, we also try increasing action diversity by adding programmatic data involving underrepresented actions such as LOCALIZEOBJECTS, ESTIMATEOBJECTDEPTH and ESTIMATEREGIONDEPTH. We experiment with data mixtures with different ratios of model-generated vs. program-generated data, ranging from 1:1 to 1:0.1.

While the best data recipe varies with the benchmark, one recipe stands out, resulting in consistent gains in the average performance across 8 benchmarks for all 3 models. This dataset consists of 293K CoTA examples generated by GPT-4o across all 31 data sources.

C.4 MODEL-BASED DATA GENERATION

C.4.1 GENERATION PROMPT

We present the full data generation prompt used in our model-based data generation pipeline in Listing 2.

C.4.2 DATASET STATISTICS

We present a table with detailed statistics of the CoTA 293K dataset in Table 6.

C.5 PROGRAMMATIC DATA GENERATION

C.5.1 ADDITIONAL DETAILS

To generate CoTA data, we define templates for thoughts, actions, and observations across all steps. In particular, we manually design five thought templates for each action and randomly sample one during generation. As for the actions, we manually select the tools for each type of questions (e.g. ESTIMATEOBJECTDEPTH for questions on objects' relative depths, and LOCALIZE for object counting questions etc.) and compose templates with them. We fill in the actions' inputs in the templates with annotations obtained from the previous step. Similarly, we define observation templates based on the action outputs' formats, and fill them in with dense annotations.

C.5.2 QA AND ACTION TEMPLATES

We present the question-answer and corresponding action templates used in our programatic data generation in Table 9. We design 16 different question templates for both single-image and multi-image examples that cover 5 capabilities: attribute recognition, counting, 2D and 3D spatial understanding, and multi-image understanding.

Table 6: **Detailed statistics of the CoTA 293K dataset.**

Original Dataset	Subset	Instance #	Max Image #	Avg Image #	Max Turn #	Avg Turn #
Cauldron	ai2d	694	2	1	6	2
	aokvqa	6322	5	1.1	7	2.1
	chartqa	4726	2	1	10	2.1
	cocoqa	13289	3	1.1	4	2
	dvqa	2158	2	1	7	2.5
	iconqa	3791	3	1.1	5	2.2
	infographic_vqa	3822	3	1	9	2.3
	mimic_cgd	6899	6	2.1	7	2.8
	nlvr2	9716	4	2.1	6	2.5
	ocrvqa	22991	2	1	7	2
	scienceqa	850	2	1	6	2.3
	st_vqa	11322	3	1	8	2
	tabmwp	14548	1	1	10	2.5
	tallyqa	16171	3	1.4	5	2.1
	textvqa	15475	5	1	6	2.1
	visual7w	4773	3	1.1	5	2.1
	vqarad	115	2	1	4	2.2
	vqav2	13394	5	1.2	6	2.1
	vsr	1864	2	1.2	4	2.1
Mantis	birds-to-words	742	4	2	5	2.7
	coinstruct	31773	8	2.3	8	2.2
	contrastive_caption	4296	8	3.6	6	2
	dreamsim	1738	3	3	3	2
	iconqa	6660	7	2.6	6	2.2
	imagecode	559	18	10.1	10	3.1
	lrv_multi	3401	9	3.3	6	2.2
	multi_vqa	2089	7	3.8	8	2.6
	nlvr2	5436	4	2	5	2.5
	spot-the-diff	2591	5	2.8	8	3
	nextqa	3057	15	8.2	9	2.3
	llava_665k_multi	77843	11	2.2	10	2.1
Total		293105	18	2.2	10	2.3

C.5.3 THOUGHT TEMPLATES

We also present the five thought templates in Listing 3 we define for each action, where one of them is randomly sampled and used during generation.

C.5.4 EXAMPLE ACTION DISTRIBUTION

We plot example distributions of all actions before and after adding programmatic CoTA 73K data in Figure 10.

D ADDITIONAL TRAINING DETAILS

We fine-tune models starting from checkpoints at different stages – pretrained and instruction tuned for Mantis-8B-SigLIP-LLaMA-3, and stage 1 and stage 1.5 for LLaVA-OneVision-7B – to investigate where CoTA data brings the largest gains. We adopt the hyperparameters from Liu et al. (2024a); Jiang et al. (2024) and fine-tune both the language model and the projector with learning rate $= 1e - 5$ for 1 epoch with either NVIDIA A100s 40GB or H100s 80GB. We perform hyperparameter tuning with LLaVA-OneVision-7B as its codebase supports flexible fine-tuning, and we vary tuning vision encoder vs. not, the language models' learning rate, and number of epochs.

We report additional training hyperparameters for Mantis models and LLaVA-OV in Table 7 and 8 respectively.

Table 7: **Additional training hyperparameters for Mantis-SigLIP and Mantis-CLIP.**

Name	Value
bf16	TRUE
tf32	True
per_device_train_batch_size	1
per_device_eval_batch_size	1
gradient_accumulation_steps	16
weight_decay	0
warmup_ratio	0.03
lr_scheduler_type	cosine
lora_enabled	FALSE
qlora_enabled	FALSE
max_seq_len	8192

Table 8: **Additional training hyperparameters for LLaVA-OV.**

Name	Value
bf16	TRUE
tf32	True
mm_vision_tower_lr	2.00E-06
mm_projector_type	mlp2x_gelu
mm_vision_select_layer	-2
image_aspect_ratio	anyres_max_9
image_grid_pinpoints	”(1x1),...,(6x6)”
mm_patch_merge_type	spatial_unpad
per_device_train_batch_size	1
per_device_eval_batch_size	1
gradient_accumulation_steps	16
weight_decay	0
warmup_ratio	0.03
lr_scheduler_type	cosine
model_max_length	8192

E ADDITIONAL EVALUATION DETAILS

On multiple-choice questions, our evaluation uses chatgpt-0125 for answer extraction. On free-form questions, we use an LLM judge (i.e. GPT-4-turbo) to score predicted answers between 0 and 1 compared to the groundtruth answers.

We present additional inference and evaluation details in Table 10 and the LLM judge prompts used for MMVet and MathVista from VLMEvalKit Duan et al. (2024) in Listings 4 and 5.

F LIMITATIONS AND FUTURE WORK

While there are many benefits of multi-modal action models, there are also limitations: they require customized action implementations and more inference compute and time. We have augmented our model mainly with tools useful for vision-centric QA and general reasoning. Due to the diversity of visual world, researchers might need to train new models with different actions for other applications such as web navigation or for other domains such as medical question answering. There are also limitations in our programmatic data, especially on the thoughts’ quality. We encourage future work to further improve the quality of CoTA data.

Table 9: **Templates for programmatic data generation.**

# of input images	Capabilities	Question Template	Action Template
1	Counting	How many {object} are there? Among {objects}, which is the most frequent object? Among {objects}, which object appears the least?	LocalizeObjects
	Counting, Attribute recognition	How many {attribute} {object} are there?	
	2D spatial reasoning	Among {objects}, which is on the most left side? Among {objects}, which is on the most right side? Among {objects}, which is on the most top side? Among {objects}, which is on the most bottom side?	
	3D spatial reasoning	Which of {objects} is closer? Which of {objects} is farther?	LocalizeObjects, EstimateRegionDepth x2 OR, EstimateObjectDepth x2
2-3	Multi-image understanding	Which image has {object}?	LocalizeObjects x N
	Multi-image understanding, Counting	How many {object} are in in these images?	
	Multi-image understanding, Counting	Which image has most {object}?	
	Multi-image understanding, Counting	Which image has least {object}?	
	Multi-image understanding, Attribute recognition	Which image has {attribute} {object}?	
	Multi-image understanding, Attribute recognition, Counting	How many {attribute} {object} in these images?	

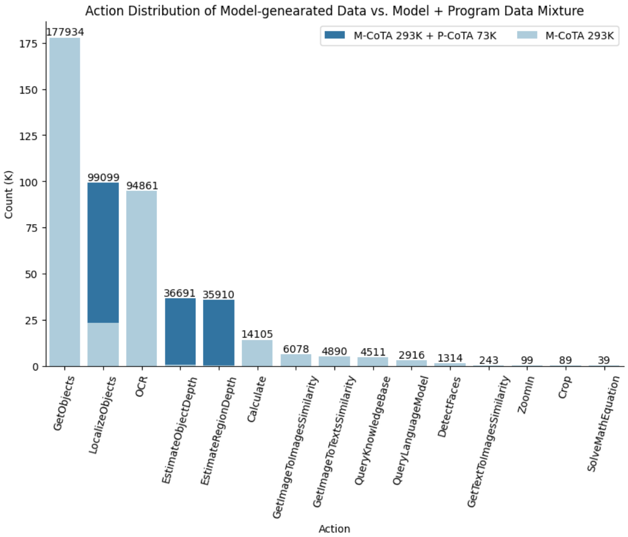


Figure 10: Action distribution of model-generated data vs. model and program data mixtures.

Table 10: **Additional inference and evaluation details.**

Stage	Name	Value
Inference	do_sample	FALSE
	temperature	0
	max_new_tokens	2000
	max_consecutive_auto_reply	10
Evaluation	llm judge for multiple choice & yes/no questions	gpt-3.5-turbo-0125
	llm judge for short answer questions (i.e. MMVet, MathVista)	gpt-4-1106-preview
	llm judge max_new_tokens	2048
	llm judge retry	5

```python
class BaseAction:
    """
    This is the Action class for agent to use.
    Using this Action class to wrap APIs, tools, models as an Action of an agent
    """

    def __init__(
        self,
        id: int,
        description: str = "",
        args_spec: dict = {},
        rets_spec: dict = {},
        examples: List = []
    ) -> None:
        """
        the agent action should be connected with data and env
        Args:
            id: the id of the action
            description: the description of the action
            args_spec: the specification of the arguments
            rets_spec: the specification of the returns
            examples: a list of examples of the action
        """
        self.name = self.__class__.__name__
        self.id = id
        self.description = description
        self.args_spec = args_spec
        self.rets_spec = rets_spec
        self.examples = examples
        self.device = "cuda:0" if torch.cuda.is_available() else "cpu"

    def __call__(self, **kwargs) -> str:
        """
        implement the Action as
        """
        raise NotImplementedError


class OCR(BaseAction):
    def __init__(self, id) -> None:
        description = "Extract texts from an image or return an empty string if no text is in the image. Note that the texts extracted may be incorrect or in the wrong order. It should be used as a reference only."
        args_spec = {"image": "the image to extract texts from."}
        rets_spec = {"text": "the texts extracted from the image."}
        examples = [{"name": "OCR", "arguments": {"image": "image-0"}}]

        super().__init__(
            id=id, description=description, args_spec=args_spec, rets_spec=rets_spec, examples=examples
        )

    def __call__(self, image, tool_version=LATEST_GPT_MODEL_ID):
        if tool_version == "easyocr":
            import easyocr
            import io
            reader = easyocr.Reader(["en"])  # Load the OCR model into memory
            image = image_processing(image)
            if isinstance(image, str):
                # If image is a path, use it directly
                image_path_or_bytes = (
                    image if os.path.exists(image) else get_full_path_data(image)
                )
            else:
                # If image is an Image object, convert it to a bytes stream
                buffer = io.BytesIO()
                image.save(buffer, format="JPEG")
                buffer.seek(0)
                image_path_or_bytes = buffer

            result = reader.readtext(image_path_or_bytes)
            result_text = [text for _, text, _ in result]
            result_formatted = {"text": ", ".join(result_text)}
        else:
            from openai import OpenAI
            import base64
            client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

            def encode_image(image_path):
                with open(image_path, "rb") as image_file:
                    return base64.b64encode(image_file.read()).decode(’utf-8’)

            image_path = image_processing(image, return_path=True)
            base64_image = encode_image(image_path)

            response = client.chat.completions.create(
                model=tool_version,
                messages=[
                    {
                        "role"   : "user",
                        "content": [
                            {"type": "text", "text": f"What are the texts in the image?"},
                            {
                                "type"     : "image_url",
                                "image_url": {
                                    "url": f"data:image/jpeg;base64,{base64_image}",
                                },
                            },
                        ],
                    }
                ],
                max_tokens=300,
            )
            result_formatted = {"text": response.choices[0].message.content}

        return result_formatted


class GetObjects(BaseAction):
    def __init__(self, id) -> None:
        description = "Using this function to get objects in an image."
        args_spec = {"image": "the image to get objects from."}
        rets_spec = {"objects": "the objects detected in the image."}
        examples = [{"name": "GetObjects", "arguments": {"image": "image-0"}}]

        super().__init__(
            id=id, description=description, args_spec=args_spec, rets_spec=rets_spec, examples=examples
        )

    def __call__(self, image, tool_version="https://huggingface.co/xinyu1205/recognize-anything-plus-model/resolve/main/ram_plus_swin_large_14m.pth?download=true"):
        from ram.models import ram_plus
        from ram import get_transform, inference_ram_openset as inference

        model_path_or_url = tool_version
        image_size = 384
        transform = get_transform(image_size=image_size)

        vit_size = "swin_l"
        # load model
        model = ram_plus(pretrained=model_path_or_url,
                         image_size=image_size,
                         vit=vit_size)
        model.eval()
        model = model.to(self.device)
        image = image_processing(image)
        image = transform(image).unsqueeze(0).to(self.device)
        tags = inference(image, model)
        objs = tags.split(" | ")
        return {"objects": objs}


class VisualizeRegionsOnImage(BaseAction):
    def __init__(self, id) -> None:
        description = "Using this function to label regions on an image."
        args_spec = {"image": "the image to label.",
                     "regions": "the regions to label on the image, where each region is represented by a dictionary with the region's bounding box and label text (can be empty string).",
                     "color": "an optional argument that specifies the color of the bounding box."
                    }
        rets_spec = {"image": "the image with regions labeled."}
        examples = [
            {"name": "VisualizeRegionsOnImage", "arguments": {"image": "image-0", "regions": [{"label": "", "bbox": [0.3, 0.2, 0.5, 0.4]}]}},
            {"name": "VisualizeRegionsOnImage", "arguments": {"image": "image-0", "regions": [{"label": "cat", "bbox": [0.3, 0.2, 0.5, 0.4]}], "color": "red"}}
        ]

        super().__init__(
            id=id, description=description, args_spec=args_spec, rets_spec=rets_spec, examples=examples
        )

    def __call__(self, image, regions: List[Region], color='yellow', width=4):
        image = image_processing(image)
        text_color = 'black'
        W,H = image.size
        img1 = image.copy()
        draw = ImageDraw.Draw(img1)
        font = ImageFont.truetype('/usr/share/fonts/truetype/dejavu/DejaVuSansMono-Bold.ttf', 16)
        for i, obj in enumerate(regions):
            bbox = obj['bbox']
            bbox = bbox[0] * W, bbox[1] * H, bbox[2] * W, bbox[3] * H
            draw.rectangle(bbox, outline=color, width=width)
            x1, y1, x2, y2 = bbox
            label = obj['label'] if "label" in obj else ""
            w,h = font.getsize(label)
            if x1 + w > W or y2  +h > H:
                draw.rectangle((x1, y2 - h, x1 + w, y2), fill=color)
                draw.text((x1, y2-h),label,fill=text_color,font=font)
            else:
                draw.rectangle((x1, y2, x1 + w, y2 + h), fill=color)
                draw.text((x1, y2),label,fill=text_color,font=font)
        return {"image": img1}


class LocalizeObjects(BaseAction):
    def __init__(self, id) -> None:
        description = "Localize one or multiple objects/regions with
    bounding boxes. This tool may output objects that don't exist or miss
     objects that do. You should use the output only as weak evidence for
     reference. When answering questions about the image, you should
    double-check the detected objects. You should be especially cautious
    about the total number of regions detected, which can be more or less
     than the actual number."
        args_spec = {
            "image": "the image to localize objects/regions in.",
            "objects": "a list of object names to localize. e.g. ['dog',
    'cat', 'person']. the model might not be able to detect rare objects
    or objects with complex descriptionriptions."
        }
        rets_spec = {"image": "the image with objects localized and
    visualized on it.", "regions": "the regions of interests localized in
     the image, where each region is represented by a dictionary with the
     region's label text, bounding box and confidence score. The
    confidence score is between 0 and 1, where 1 means the model is very
    confident. Note that both the bounding boxes and confidence scores
    can be unreliable and should only be used as reference."}
        examples = [{"name": "LocalizeObjects", "arguments": {"image": "
    image-0", "objects": ["dog", "cat"]}}]

        super().__init__(
            id=id, description=description, args_spec=args_spec,
    rets_spec=rets_spec, examples=examples
        )

    def __call__(self, image, objects: List[str]):
        from groundingdino.util.inference import load_model, load_image,
    predict, annotate
        import cv2
        text = ". ".join(objects)
        model = load_model("/user/mma/mma/GroundingDINO/groundingdino/
    config/GroundingDINO_SwinT_OGC.py",
                           "/user/mma/mma/GroundingDINO/weights/
    groundingdino_swint_ogc.pth",
                           device=self.device)
        BOX_TRESHOLD = 0.35
        TEXT_TRESHOLD = 0.25
        image_path = image_processing(image, return_path=True)
        original_image = image_processing(image)
        image_source, image = load_image(image_path)

        boxes, logits, phrases = predict(
            model=model,
            image=image,
            caption=text,
            box_threshold=BOX_TRESHOLD,
            text_threshold=TEXT_TRESHOLD
        )

        objects = []
        obj_cnt = {}
```

```python
        for i in range(len(boxes)):
            xyxy = box_convert(boxes=boxes[i], in_fmt="cxcywh", out_fmt="xyxy").numpy()
            bbox = [round(val, 2) for val in list(xyxy)]
            score = round(logits[i].item(), 2)
            phrase = phrases[i]
            obj_cnt[phrase] = obj_cnt.get(phrase, 0) + 1
            phrase = f"{phrase}-{obj_cnt[phrase]}" if obj_cnt[phrase] > 1 else phrase
            objects.append({"label": phrase, "bbox": bbox, "score": score})
        visualize = VisualizeRegionsOnImage(0)
        results = visualize(image=original_image, regions=objects)
        tagged_image = results["image"]
        results_formatted = {"regions": objects, "image": tagged_image}
        return results_formatted


class Crop(BaseAction):
    def __init__(self, id) -> None:
        description = "Crop an image with the bounding box. It labels the cropped region with a bounding box and crops the region with some margins around the bounding box to help with contextual understanding of the region."
        args_spec = {
            "image": "the image to crop.",
            "bbox": "the bbox to crop. It should be a list of [left, top, right, bottom], where each value is a float between 0 and 1 to represent the percentage of the image width/height and how far it is from the top left corner at [0, 0].",
        }
        rets_spec = {"image": "the cropped image."}
        examples = [{"name": "Crop", "arguments": {"image": "image-0", "bbox": [0.33, 0.21, 0.58, 0.46]}}]

        super().__init__(
            id=id, description=description, args_spec=args_spec, rets_spec=rets_spec, examples=examples
        )

    def __call__(self, image, bbox):
        image = image_processing(image)

        if isinstance(bbox, str):
            try:
                bbox = ast.literal_eval(bbox)
            except:
                bbox = []

        use_percent = (all(x <= 1.0 for x in bbox))
        if not use_percent:
            raise ValueError("Bounding box coordinates must be between 0 and 1.")

        visualize = VisualizeRegionsOnImage(0)
        results = visualize(image=image, regions=[{"label": "", "bbox": bbox}])
        image = results["image"]

        W, H = image.size
        bbox = [bbox[0] * W, bbox[1] * H, bbox[2] * W, bbox[3] * H]
        bbox = expand_bbox(bbox, image.size)
        out_img = image.crop(bbox)
        return {"image": out_img}
```

```python

class ZoomIn(BaseAction):
    def __init__(self, id) -> None:
        description = "Zoom in on a region of the input image. This tool first crops the specified region from the image with the bounding box and then resizes the cropped region to create the zoom effect. It also adds some margins around the cropped region to help with contextual understanding of the region."
        args_spec = {
            "image": "the image to zoom in on.",
            "bbox": "The bbox should be a list of [left, top, right, bottom], where each value is a float between 0 and 1 to represent the percentage of the image width/height and how far it is from the top left corner at [0, 0].",
            "zoom_factor": "the factor to zoom in by. It should be greater than 1.",
        }
        rets_spec = {"image": "the zoomed in image."}
        examples = [
            {"name": "ZoomIn", "arguments": {"image": "image-0", "bbox": [0.4, 0.3, 0.5, 0.4], "zoom_factor": 2}},
        ]

        super().__init__(
            id=id, description=description, args_spec=args_spec, rets_spec=rets_spec, examples=examples
        )

    def __call__(self, image, bbox, zoom_factor):
        if zoom_factor <= 1:
            raise ValueError("Zoom factor must be greater than 1 to zoom in")

        image = image_processing(image)
        use_percent = (all(x <= 1.0 for x in bbox))
        if not use_percent:
            raise ValueError("Bounding box coordinates must be between 0 and 1.")

        crop = Crop(0)
        cropped_image = crop(image, bbox)["image"]

        W, H = cropped_image.size

        # Calculate the size of the zoomed image
        new_width = int(W * zoom_factor)
        new_height = int(H * zoom_factor)

        # Resize the cropped image to create the zoom effect
        zoomed_image = cropped_image.resize((new_width, new_height), Image.LANCZOS)
        return {’image’: zoomed_image}


class GetImageToImagesSimilarity(BaseAction):
    def __init__(self, id) -> None:
        description = "Get the similarity between one image and a list of other images. Note that this similarity score may not be accurate and should be used as a reference only."
        args_spec = {
            "image": "the reference image.",
            "other_images": "the other images to compare to the reference image.",
        }
        rets_spec = {"similarity": "the CLIP similarity scores between the reference image and the other images.", "best_image_index": "the index of the most similar image."}
        examples = [
            {"name": "GetImageToImagesSimilarity", "arguments": {"image": "image-0", "other_images": ["image-1", "image-2"]}}
        ]

        super().__init__(
            id=id, description=description, args_spec=args_spec, rets_spec=rets_spec, examples=examples
        )

    def __call__(self, image, other_images, tool_version=’ViT-H-14-378-quickgelu’, other_images_raw=None):
        import torch
        import open_clip
        original_images = other_images_raw
        model, _, preprocess = open_clip.create_model_and_transforms(tool_version, pretrained=’dfn5b’)
        model.eval()
        image = image_processing(image)
        images = [image_processing(image) for image in other_images]

        image = preprocess(image).unsqueeze(0)
        images = torch.stack([preprocess(image) for image in images])

        with torch.no_grad(), torch.cuda.amp.autocast():
            image1_features = model.encode_image(image)
            image2_features = model.encode_image(images)

            image1_features /= image1_features.norm(dim=-1, keepdim=True)
            image2_features /= image2_features.norm(dim=-1, keepdim=True)

            probs = image1_features @ image2_features.T
        sim_scores = [round(sim_score, 2) for sim_score in probs[0].tolist()]
        best_image_match = torch.argmax(probs).item()
        return {’similarity’: sim_scores, "best_image_index": best_image_match, "best_image": original_images[best_image_match]}


class GetImageToTextsSimilarity(BaseAction):
    def __init__(self, id) -> None:
        description = "Get the similarity between one image and a list of texts. Note that this similarity score may not be accurate and should be used as a reference only."
        args_spec = {
            "image": "the reference image.",
            "texts": "a list of texts to compare to the reference image.",
        }
        rets_spec = {"similarity": "the CLIP similarity between the image and the texts.", "best_text_index": "the index of the most similar text.", "best_text": "the most similar text."}
        examples = [
            {"name": "GetImageToTextsSimilarity", "arguments": {"image": "image-0", "texts": ["a cat", "a dog"]}}
        ]

        super().__init__(
            id=id, description=description, args_spec=args_spec, rets_spec=rets_spec, examples=examples
        )

    def __call__(self, image, texts, tool_version='ViT-H-14-378-quickgelu'):
        import torch
        import open_clip

        model, _, preprocess = open_clip.create_model_and_transforms(tool_version, pretrained='dfn5b')
        model.eval()  # model in train mode by default, impacts some models with BatchNorm or stochastic depth active
        tokenizer = open_clip.get_tokenizer(tool_version)

        image = preprocess(image_processing(image)).unsqueeze(0)
        text = tokenizer(texts)

        with torch.no_grad(), torch.cuda.amp.autocast():
            image_features = model.encode_image(image)
            text_features = model.encode_text(text)
            image_features /= image_features.norm(dim=-1, keepdim=True)
            text_features /= text_features.norm(dim=-1, keepdim=True)

            probs = image_features @ text_features.T
        sim_scores = [round(sim_score, 2) for sim_score in probs[0].tolist()]
        best_text_match = torch.argmax(probs).item()
        return {'similarity': sim_scores, "best_text_index": best_text_match, "best_text": texts[best_text_match]}


class GetTextToImagesSimilarity(BaseAction):
    def __init__(self, id) -> None:
        description = "Get the similarity between one text and a list of images. Note that this similarity score may not be accurate and should be used as a reference only."
        args_spec = {
            "text": "the reference text.",
            "images": "a list of images to compare to the reference text.",
        }
        rets_spec = {"similarity": "the CLIP similarity between the image and the texts.", "best_image_index": "the index of the most similar image."}
        examples = [
            {"name": "GetTextToImagesSimilarity", "arguments": {"text": "a black and white cat", "images": ["image-0", "image-1"]}}
        ]

        super().__init__(
            id=id, description=description, args_spec=args_spec, rets_spec=rets_spec, examples=examples
        )

    def __call__(self, text, images, tool_version='ViT-H-14-378-quickgelu'):
        import torch
        import open_clip
        original_images = images
        model, _, preprocess = open_clip.create_model_and_transforms(tool_version, pretrained='dfn5b')
        model.eval()  # model in train mode by default, impacts some models with BatchNorm or stochastic depth active
        tokenizer = open_clip.get_tokenizer(tool_version)

        text = tokenizer([text])
        images = [image_processing(image) for image in images]
        images = torch.stack([preprocess(image) for image in images])

        with torch.no_grad(), torch.cuda.amp.autocast():
            image_features = model.encode_image(images)
            text_features = model.encode_text(text)
            image_features /= image_features.norm(dim=-1, keepdim=True)
            text_features /= text_features.norm(dim=-1, keepdim=True)

            probs = text_features @ image_features.T
        sim_scores = [round(sim_score, 2) for sim_score in probs[0].
    tolist()]
        best_image_match = torch.argmax(probs).item()
        return {'similarity': sim_scores, "best_image_index":
    best_image_match, "best_image": original_images[best_image_match]}


class EstimateObjectDepth(BaseAction):
    def __init__(self, id) -> None:
        description = "Estimate the depth of an object in an image using
    DepthAnything model. It returns an estimated depth value of the
    object specified by the a brief text description. The smaller the
    value is, the closer the object is to the camera, and the larger the
    farther. This tool may help you to better reason about the spatial
    relationship, like which object is closer to the camera."
        args_spec = {
            "image": "the image to get the depth from.",
            "object": "a short description of the object to get the depth
     from.",
        }
        rets_spec = {"depth": "the estimated depth of the object."}
        examples = [
            {"name": "EstimateObjectDepth", "arguments": {"image": "image
    -0", "object": "a black cat"}},
        ]

        super().__init__(
            id=id, description=description, args_spec=args_spec,
    rets_spec=rets_spec, examples=examples
        )

    def __call__(self, image, object, mode="mean"):
        action = LocalizeObjects(0)
        results = action(image=image, objects=[object])
        if len(results["regions"]) == 0:
            return {"depth": "Object not found."}
        else:
            # use the best match object's bbox
            best_match = np.argmax([region["score"] for region in results
    ["regions"]])
            bbox = results["regions"][best_match]["bbox"]
            depth_estimator = EstimateRegionDepth(0)
            return depth_estimator(image=image, bbox=bbox, mode=mode)


class EstimateRegionDepth(BaseAction):
    def __init__(self, id) -> None:
        description = "Estimate the depth of a region in an image using
    DepthAnything model. It returns an estimated depth value of the
    region specified by the input bounding box. The smaller the value is,
     the closer the region is to the camera, and the larger the farther.
    This tool may help you to better reason about the spatial
    relationship, like which object is closer to the camera. "
        args_spec = {
            "image": "the image to get the depth from.",
            "bbox": "the bbox of the region to get the depth from. It
    should be a list of [left, top, right, bottom], where each value is a
```

```
 float between 0 and 1 to represent the percentage of the image width/height and how far it is from the top left corner at [0, 0].",
            # "mode": "the mode to get the depth. It should be one of ’center’ or ’average’. ’center’ returns the depth of the center of the region. ’average’ returns the average depth of the region.",
        }
        rets_spec = {"depth": "the estimated depth of the region."}
        examples = [
            {"name": "EstimateRegionDepth", "arguments": {"image": "image-0", "bbox": [0.3, 0.2, 0.5, 0.4]}},
        ]
        super().__init__(
            id=id, description=description, args_spec=args_spec, rets_spec=rets_spec, examples=examples
        )

    def __call__(self, image, bbox: List[str], mode="mean"):
        import numpy as np
        from scipy import stats
        image = image_processing(image)
        depth_model = pipeline(task="depth-estimation", model="depth-anything/Depth-Anything-V2-Small-hf", device=self.device)
        result = depth_model(image)
        depth = result["predicted_depth"][0].numpy()
        depth = depth.max() - depth # smaller values in depth map are farther from the camera so reversing the values
        H, W = depth.shape

        use_percent = all(x <= 1.0 for x in bbox)
        if not use_percent:
            raise ValueError("Bounding box coordinates must be between 0 and 1.")
        bbox = [bbox[0] * W, bbox[1] * H, bbox[2] * W, bbox[3] * H]
        if mode == "center":
            x, y = (bbox[0] + bbox[2]) / 2, (bbox[1] + bbox[3]) / 2
            x, y = int(x), int(y)
            depth_value = depth[y, x]
        elif mode == "mean":
            x1, y1, x2, y2 = bbox
            x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
            depth_value = np.mean(depth[y1:y2, x1:x2])
        elif mode == "mode":
            x1, y1, x2, y2 = bbox
            x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
            mode_result = stats.mode(depth[y1:y2, x1:x2])
            depth_value = mode_result.mode[0]
        else:
            raise NotImplementedError(f"Depth mode {mode} is not supported.")
        return {"depth": round(depth_value, 2)}


class Calculate(BaseAction):
    def __init__(self, id) -> None:
        description = "Calculate a math expression."
        args_spec = {"expression": "the math expression to calculate."}
        rets_spec = {"result": "the result of the math expression."}
        examples = [
            {"name": "Calculate", "arguments": {"expression": "2 + 2"}},
            {"name": "Calculate", "arguments": {"expression": "4*9*84"}},
            {"name": "Calculate", "arguments": {"expression": "5-4/2"}},
        ]

        super().__init__(
            id=id, description=description, args_spec=args_spec, rets_spec=rets_spec, examples=examples
        )

    def __call__(self, expression):
        result = eval(expression)
        return {"result": result}


class SolveMathEquation(BaseAction):
    def __init__(self, id) -> None:
        description = "Using this action to solve a math problem with WolframAlpha."
        args_spec = {"query": "a question that involves a math equation to be solved."}
        rets_spec = {"result": "the result of the query."}
        examples = [
            {"name": "SolveMathEquation", "arguments": {"query": "2 + 2=?"}},
            {"name": "SolveMathEquation", "arguments": {"query": "x^2 + 2x + 1 = 0, what is x?"}},
        ]

        self.client = wolframalpha.Client(os.getenv("WOLFRAM_ALPHA_API_KEY"))
        super().__init__(
            id=id, description=description, args_spec=args_spec, rets_spec=rets_spec, examples=examples
        )

    def __call__(self, query):
        from urllib.error import HTTPError

        is_success = False

        res = self.client.query(query)

        if not res["@success"]:
            return (
                "Your Wolfram query is invalid. Please try a new query for wolfram.",
                is_success,
            )
        assumption = next(res.pods).text
        answer = ""
        for result in res["pod"]:
            if result["@title"] == "Solution":
                answer = result["subpod"]["plaintext"]
            if result["@title"] == "Results" or result["@title"] == "Solutions":
                for i, sub in enumerate(result["subpod"]):
                    answer += f"ans {i}: " + sub["plaintext"] + "\n"
                break
        if answer == "":
            answer = next(res.results).text

        if answer is None or answer == "":
            return {"result": "No good Wolfram Alpha Result was found"}
        else:
            return {"result": answer}


class DetectFaces(BaseAction):
    def __init__(self, id) -> None:
        description = "Using this function to detect faces in an image."
        args_spec = {"image": "the image to detect faces from."}
        rets_spec = {"image": "the image with objects localized and visualized on it.", "regions": "the regions of the faces detected, where each regin is represented by a dictionary with the region's label text and bounding box."}
        examples = [
            {"name": "DetectFaces", "arguments": {"image": "image-0"}}
        ]
        import face_detection
        ckpt_path = f"/root/.cache/torch/hub/checkpoints/WIDERFace_DSFD_RES152.pth"
        if not os.path.exists(ckpt_path):
            from huggingface_hub import hf_hub_download
            hf_hub_download(repo_id="user/mma", filename="WIDERFace_DSFD_RES152.pth", local_dir="/root/.cache/torch/hub/checkpoints/")

        self.model = face_detection.build_detector(
            "DSFDDetector", confidence_threshold=.5, nms_iou_threshold=.3)
        super().__init__(
            id=id, description=description, args_spec=args_spec, rets_spec=rets_spec, examples=examples
        )

    def enlarge_face(self,box,W,H,f=1.5):
        x1,y1,x2,y2 = box
        w = int((f-1)*(x2-x1)/2)
        h = int((f-1)*(y2-y1)/2)
        x1 = max(0,x1-w)
        y1 = max(0,y1-h)
        x2 = min(W,x2+w)
        y2 = min(H,y2+h)
        return [x1,y1,x2,y2]

    def __call__(self, image):
        import numpy as np
        image = image_processing(image)

        with torch.no_grad():
            faces = self.model.detect(np.array(image))

        W,H = image.size
        objs = []
        for i,box in enumerate(faces):
            x1,y1,x2,y2,c = [int(v) for v in box.tolist()]
            normalized_bbox = [x1/W, y1/H, x2/W, y2/H]
            objs.append(dict(
                bbox=[round(num, 2) for num in normalized_bbox],
                label=f'face {i+1}' if i > 0 else 'face',
            ))
        visualize = VisualizeRegionsOnImage(0)
        results = visualize(image=image, regions=objs)
        tagged_image = results["image"]
        results_formatted = {"regions": objs, "image": tagged_image}
        return results_formatted


class QueryLanguageModel(BaseAction):
    def __init__(self, id) -> None:
        description = "Using this function to ask a language model a question."
        args_spec = {"query": "the question to ask the language model."}
        rets_spec = {"result": "the response from the language model."}
        examples = [
            {"name": "QueryLanguageModel", "arguments": {"query": "What is the capital of France?"}},
        ]
        super().__init__(
            id=id, description=description, args_spec=args_spec, rets_spec=rets_spec, examples=examples
        )

    def __call__(self, query):
        from openai import OpenAI
        client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

        response = client.chat.completions.create(
            model=LATEST_GPT_MODEL_ID,
            messages=[
                {
                    "role"   : "user",
                    "content": [
                        {"type": "text", "text": f"{query}"},
                    ],
                }
            ],
            max_tokens=300,
        )

        return {’result’: response.choices[0].message.content}


class QueryKnowledgeBase(BaseAction):
    def __init__(self, id) -> None:
        description = "Using this function to query a knowledge base."
        args_spec = {"query": "the query to search in a knowledge base such as wikipedia."}
        rets_spec = {"result": "the answer from the knowledge base."}
        examples = [
            {"name": "QueryKnowledgeBase", "arguments": {"query": "Paris"}},
        ]

        super().__init__(
            id=id, description=description, args_spec=args_spec, rets_spec=rets_spec, examples=examples
        )

    def __call__(self, query, lang="en", sentences=2, knowledge_base="wikipedia"):
        if knowledge_base == "wikipedia":
            # Set the language for Wikipedia (default is ’en’ for English)
            wikipedia.set_lang(lang)

            # Search Wikipedia for pages related to the query
            search_results = wikipedia.search(query)
            if not search_results:
                return {"No results found."}

            # Get the summary of the first search result
            page = wikipedia.page(search_results[0])
            summary = wikipedia.summary(page.title, sentences=sentences)
            result = {
                "title": page.title,
                "url": page.url,
                "summary": summary
            }
            return result
        else:
            raise NotImplementedError(f"Knowledge base {knowledge_base} is not supported.")


class Terminate(BaseAction):
    def __init__(self, id) -> None:
        description = "Using this function to finish the task."
        args_spec = {"answer": "the final answer."}
        rets_spec = {"answer": "the final answer."}
        examples = [{"name": "Terminate", "arguments": {"answer": "yes"}}]

        super().__init__(
            id=id, description=description, args_spec=args_spec, rets_spec=rets_spec, examples=examples
        )

    def __call__(self, answer):
        return {"answer": answer}
```

Listing 1: Python implementation of all actions

```
[BEGIN OF GOAL]
You are a helpful assistant, and your goal is to solve the # USER REQUEST #. You can either rely on your own capabilities or perform actions with external tools to help you. A list of all available actions are provided to you in the below.
[END OF GOAL]

[BEGIN OF ACTIONS]
Name: OCR
Description: Extract texts from an image or return an empty string if no text is in the image. Note that the texts extracted may be incorrect or in the wrong order. It should be used as a reference only.
Arguments: {’image’: ’the image to extract texts from.’}
Returns: {’text’: ’the texts extracted from the image.’}
Examples:
{"name": "OCR", "arguments": {"image": "image-0"}}

Name: LocalizeObjects
Description: Localize one or multiple objects/regions with bounding boxes. This tool may output objects that don’t exist or miss objects that do. You should use the output only as weak evidence for reference. When answering questions about the image, you should double-check the detected objects. You should be especially cautious about the total number of regions detected, which can be more or less than the actual number.
Arguments: {’image’: ’the image to localize objects/regions in.’, ’objects’: "a list of object names to localize. e.g. [’dog’, ’cat’, ’person’]. the model might not be able to detect rare objects or objects with complex descriptionriptions."}
Returns: {’image’: ’the image with objects localized and visualized on it.’, ’regions’: "the regions of interests localized in the image, where each region is represented by a dictionary with the region’s label text, bounding box and confidence score. The confidence score is between 0 and 1, where 1 means the model is very confident. Note that both the bounding boxes and confidence scores can be unreliable and should only be used as reference."}
Examples:
{"name": "LocalizeObjects", "arguments": {"image": "image-0", "objects": ["dog", "cat"]}}

Name: GetObjects
Description: Using this function to get objects in an image.
Arguments: {’image’: ’the image to get objects from.’}
Returns: {’objects’: ’the objects detected in the image.’}
Examples:
{"name": "GetObjects", "arguments": {"image": "image-0"}}

Name: EstimateRegionDepth
Description: Estimate the depth of a region in an image using DepthAnything model. It returns an estimated depth value of the region specified by the input bounding box. The smaller the value is, the closer the region is to the camera, and the larger the farther. This tool may help you to better reason about the spatial relationship, like which object is closer to the camera.
Arguments: {’image’: ’the image to get the depth from.’, ’bbox’: ’the bbox of the region to get the depth from. It should be a list of [left, top, right, bottom], where each value is a float between 0 and 1 to represent the percentage of the image width/height and how far it is from the top left corner at [0, 0].’}
Returns: {’depth’: ’the estimated depth of the region.’}
Examples:
{"name": "EstimateRegionDepth", "arguments": {"image": "image-0", "bbox": [0.3, 0.2, 0.5, 0.4]}}

Name: EstimateObjectDepth
```

```
Description: Estimate the depth of an object in an image using DepthAnything model. It returns an estimated depth value of the object specified by the a brief text description. The smaller the value is, the closer the object is to the camera, and the larger the farther. This tool may help you to better reason about the spatial relationship, like which object is closer to the camera.
Arguments: {’image’: ’the image to get the depth from.’, ’object’: ’a short description of the object to get the depth from.’}
Returns: {’depth’: ’the estimated depth of the object.’}
Examples:
{"name": "EstimateObjectDepth", "arguments": {"image": "image-0", "object": "a black cat"}}

Name: Crop
Description: Crop an image with the bounding box. It labels the cropped region with a bounding box and crops the region with some margins around the bounding box to help with contextual understanding of the region.
Arguments: {’image’: ’the image to crop.’, ’bbox’: ’the bbox to crop. It should be a list of [left, top, right, bottom], where each value is a float between 0 and 1 to represent the percentage of the image width/height and how far it is from the top left corner at [0, 0].’}
Returns: {’image’: ’the cropped image.’}
Examples:
{"name": "Crop", "arguments": {"image": "image-0", "bbox": [0.33, 0.21, 0.58, 0.46]}}

Name: ZoomIn
Description: Zoom in on a region of the input image. This tool first crops the specified region from the image with the bounding box and then resizes the cropped region to create the zoom effect. It also adds some margins around the cropped region to help with contextual understanding of the region.
Arguments: {’image’: ’the image to zoom in on.’, ’bbox’: ’The bbox should be a list of [left, top, right, bottom], where each value is a float between 0 and 1 to represent the percentage of the image width/height and how far it is from the top left corner at [0, 0].’, ’zoom_factor’: ’the factor to zoom in by. It should be greater than 1.’}
Returns: {’image’: ’the zoomed in image.’}
Examples:
{"name": "ZoomIn", "arguments": {"image": "image-0", "bbox": [0.4, 0.3, 0.5, 0.4], "zoom_factor": 2}}

Name: QueryLanguageModel
Description: Using this function to ask a language model a question.
Arguments: {’query’: ’the question to ask the language model.’}
Returns: {’result’: ’the response from the language model.’}
Examples:
{"name": "QueryLanguageModel", "arguments": {"query": "What is the capital of France?"}}

Name: GetImageToImagesSimilarity
Description: Get the similarity between one image and a list of other images. Note that this similarity score may not be accurate and should be used as a reference only.
Arguments: {’image’: ’the reference image.’, ’other_images’: ’the other images to compare to the reference image.’}
Returns: {’similarity’: ’the CLIP similarity scores between the reference image and the other images.’, ’best_image_index’: ’the index of the most similar image.’}
Examples:
{"name": "GetImageToImagesSimilarity", "arguments": {"image": "image-0", "other_images": ["image-1", "image-2"]}}

```

```
Name: GetImageToTextsSimilarity
Description: Get the similarity between one image and a list of texts. Note that this similarity score may not be accurate and should be used as a reference only.
Arguments: {’image’: ’the reference image.’, ’texts’: ’a list of texts to compare to the reference image.’}
Returns: {’similarity’: ’the CLIP similarity between the image and the texts.’, ’best_text_index’: ’the index of the most similar text.’, ’best_text’: ’the most similar text.’}
Examples:
{"name": "GetImageToTextsSimilarity", "arguments": {"image": "image-0", "texts": ["a cat", "a dog"]}}

Name: GetTextToImagesSimilarity
Description: Get the similarity between one text and a list of images. Note that this similarity score may not be accurate and should be used as a reference only.
Arguments: {’text’: ’the reference text.’, ’images’: ’a list of images to compare to the reference text.’}
Returns: {’similarity’: ’the CLIP similarity between the image and the texts.’, ’best_image_index’: ’the index of the most similar image.’}
Examples:
{"name": "GetTextToImagesSimilarity", "arguments": {"text": "a black and white cat", "images": ["image-0", "image-1"]}}

Name: DetectFaces
Description: Using this function to detect faces in an image.
Arguments: {’image’: ’the image to detect faces from.’}
Returns: {’image’: ’the image with objects localized and visualized on it.’, ’regions’: "the regions of the faces detected, where each regin is represented by a dictionary with the region’s label text and bounding box."}
Examples:
{"name": "DetectFaces", "arguments": {"image": "image-0"}}

Name: QueryKnowledgeBase
Description: Using this function to query a knowledge base.
Arguments: {’query’: ’the query to search in a knowledge base such as wikipedia.’}
Returns: {’result’: ’the answer from the knowledge base.’}
Examples:
{"name": "QueryKnowledgeBase", "arguments": {"query": "Paris"}}

Name: Calculate
Description: Calculate a math expression.
Arguments: {’expression’: ’the math expression to calculate.’}
Returns: {’result’: ’the result of the math expression.’}
Examples:
{"name": "Calculate", "arguments": {"expression": "2 + 2"}}
{"name": "Calculate", "arguments": {"expression": "4*9*84"}}
{"name": "Calculate", "arguments": {"expression": "5-4/2"}}

Name: SolveMathEquation
Description: Using this action to solve a math problem with WolframAlpha.
Arguments: {’query’: ’a question that involves a math equation to be solved.’}
Returns: {’result’: ’the result of the query.’}
Examples:
{"name": "SolveMathEquation", "arguments": {"query": "2 + 2=?"}}
{"name": "SolveMathEquation", "arguments": {"query": "x^2 + 2x + 1 = 0, what is x?"}}

Name: Terminate
Description: Using this function to finish the task.
Arguments: {’answer’: ’the final answer.’}
```

```
Returns: {'answer': 'the final answer.'}
Examples:
{"name": "Terminate", "arguments": {"answer": "yes"}}

[END OF ACTIONS]

[BEGIN OF TASK INSTRUCTIONS]
1. You must only select actions from # ACTIONS #.
2. You can only call one action at a time.
3. If no action is needed, please make actions an empty list (i.e. ''
    actions'': []).
4. You must always call Terminate with your final answer at the end.
[END OF TASK INSTRUCTIONS]

[BEGIN OF FORMAT INSTRUCTIONS]
Your output should be in a strict JSON format as follows:
{"thought": "the thought process, or an empty string", "actions": [{"name
    ": "action1", "arguments": {"argument1": "value1", "argument2": "
    value2"}}]}
[END OF FORMAT INSTRUCTIONS]

[BEGIN OF EXAMPLES]:
# USER REQUEST #:
 In image-0, Which of the two objects on the plate is the biggest?
A. The pile of scrambled eggs is the biggest.
B. The strawberries are the biggest object.
Please answer directly with only the letter of the correct option and
    nothing else.
# RESPONSE #:
{"thought": "To determine which of the two objects on the plate is larger
    , I need to analyze the size of the scrambled eggs, and the
    strawberries", "actions": [{"name": "LocalizeObjects", "arguments":
    {"image": "image-0", "objects": ["scrambled eggs", "strawberries
    "]}}]}
OBSERVATION:
{"image": "image-1", "regions": [{"label": "eggs", "bbox": [0.5, 0.6,
    0.6, 0.8], "score": 0.85}, {"label": "strawberries", "bbox": [0.4,
    0.5, 0.45, 0.7], "score": 0.54}]}
{"thought": "To calculate the area of a bounding box, we can use the
    formula: area = (x_max - x_min) * (y_max - y_min). We first get the
    area of the scrambled eggs.", "actions": [{"name": "Calculate", "
    arguments": {"expression": "(0.6-0.5) * (0.8-0.6)"}}]}
OBSERVATION:
{"result": "0.02"}
{"thought": "Then, we also calculate the area of the strawberries.", "
    actions": [{"name": "Calculate", "arguments": {"expression":
    "(0.45-0.4) * (0.7-0.5)"}}]}
OBSERVATION:
{"result": "0.01"}
{"thought": "Since 0.02 > 0.01, it is apparent that the eggs cover a
    larger area within their bounding box.", "actions": [{"name": "
    Terminate", "arguments": {"answer": "A"}}]}

# USER REQUEST #:
 Given the input image image-0, How many pedestrians are there in the
    image? Please answer directly with a single word or number.
# RESPONSE #:
{"thought": "To determine the number of pedestrians, I need to first
    localize them on the image.", "actions": [{"name": "LocalizeObjects",
     "arguments": {"image": "image-0", "objects": ["person"]}}]}
OBSERVATION:
{"image": "image-1", "regions": [{"label": "person", "bbox": [0.77, 0.47,
     0.79, 0.54], "score": 0.83}, {"label": "person-2", "bbox": [0.69,
    0.49, 0.7, 0.52], "score": 0.43}]}
```

```
{"thought": "The LocalizeObjects action returns two regions for \"person\", but one of the regions has a lower confidence score. Upon a closer look at the output image image-1, we can see that there is actually only one pedestrian in the image.", "actions": [{"name": "Terminate", "arguments": {"answer": "1"}}]}

# USER REQUEST #:
 Based on image-0, is the object on top bigger than the object below?
A. The object on the bottom is bigger.
B. The object on top is bigger.
C. Both objects are the same size.
Please answer directly with only the letter of the correct option and nothing else.
# RESPONSE #:
{"thought": "By looking at the image, we can see that both objects are game consoles of the same brand and size.", "actions": [{"name": "Terminate", "arguments": {"answer": "C"}}]}

# USER REQUEST #:
 What is x in the image?
# RESPONSE #:
{"thought": "To get the result of the equation, I need to first extract the equation from the image.", "actions": [{"name": "OCR", "arguments": {"image": "image-0"}}]}
OBSERVATION:
{"text": "x-2^3=0"}
{"thought": "The math equation is ’x-2^3=0’, and I need to find x. I can solve it with a math-related tool.", "actions": [{"name": "SolveMathEquation", "arguments": {"query": "x-2^3=0, what is x?"}}]}
OBSERVATION:
{"result": "x = 8"}
{"thought": "As suggested in the last observation, the answer is 8.", "actions": [{"name": "Terminate", "arguments": {"answer": "8"}}]}

[END OF EXAMPLES]
```

Listing 2: Model-based data generation system prompt

```
def GetObjects_template():
    thought_templates = ["I need to check what objects are present in the {image_kw}.",
                        "I need to analyze the {image_kw} for context."
                        "I need to identify the objects in the {image_kw}.",
                        "To answer the question, let’s first analyze the {image_kw}.",
                        "To answer the question, analyzing the objects in the {image_kw} is necessary."]
    return thought_templates

def LocalizeObjects_template():
    thought_templates = ["I need to analyze the positions of {objects} in the {image_kw}.",
                        "I need to analyze the locations of {objects} in the {image_kw}.",
                        "I need to localize the {objects} based on the {image_kw}.",
                        "I’ll identify the positions of {objects} in the {image_kw}.",
                        "I need to determine the positions of {objects} by analyzing the {image_kw}."]
    return thought_templates

def EstimateObjectDepth_template():
    thought_templates = ["I should estimate the depth of {object} to determine whether it is closer or farther.",
                         "I will estimate the depth of {object}.",
                         "I need to estimate the depth for {object} to make a comparison.",
                         "To determine how far {object} is, I need to evaluate the distance to it.",
                         "I now need to estimate the depth for {object}."]
    return thought_templates


def EstimateRegionDepth_template():
    thought_templates = ["I should estimate the objects’ depths to determine which one is closer.",
                         "I need to estimate the region’s depth in the image.",
                         "I need to determine the depths of the detected objects based on their positions.",
                         "I need to estimate the depth of the objects to make a comparison.",
                         "To determine the relative proximity of the objects in the image, I need to estimate the depth of each object."]
    return thought_templates

def Terminate_template():
    thought_templates = ["Based on the information above, I can conclude that the answer is {answer}",
                         "Based on a close analysis of the {image_kw} and additional information above, I believe the answer is {answer}.",
                         "I have analyzed the {image_kw} and the information above, and I believe the answer is {answer}.",
                         "The {image_kw} and the information above suggest that the answer is {answer}.",
                         "According to the content of the {image_kw} and the observations, I can conclude that the answer is {answer}."]
    return thought_templates
```

Listing 3: Thought templates for each action

```
Compare the ground truth and prediction from AI models, to give a correctness score for the prediction. <AND> in the ground truth means it is totally right only when all elements in the ground truth are present in the prediction, and <OR> means it is totally right when any one element in the ground truth is present in the prediction. The correctness score is 0.0 (totally wrong), 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, or 1.0 (totally right). Just complete the last space of the correctness score.
Question | Ground truth | Prediction | Correctness
--- | --- | --- | ---
What is x in the equation? | -1 <AND> -5 | x = 3 | 0.0
What is x in the equation? | -1 <AND> -5 | x = -1 | 0.5
What is x in the equation? | -1 <AND> -5 | x = -5 | 0.5
What is x in the equation? | -1 <AND> -5 | x = -5 or 5 | 0.5
What is x in the equation? | -1 <AND> -5 | x = -1 or x = -5 | 1.0
Can you explain this meme? | This meme is poking fun at the fact that the names of the countries Iceland and Greenland are misleading. Despite its name, Iceland is known for its beautiful green landscapes, while Greenland is mostly covered in ice and snow. The meme is saying that the person has trust issues because the names of these countries do not accurately represent their landscapes. | The meme talks about Iceland and Greenland. It’s pointing out that despite their names, Iceland is not very icy and Greenland isn’t very green. | 0.4
Can you explain this meme? | This meme is poking fun at the fact that the names of the countries Iceland and Greenland are misleading. Despite its name, Iceland is known for its beautiful green landscapes, while Greenland is mostly covered in ice and snow. The meme is saying that the person has trust issues because the names of these countries do not accurately represent their landscapes. | The meme is using humor to point out the misleading nature of Iceland’s and Greenland’s names.
Iceland, despite its name, has lush green landscapes while Greenland is mostly covered in ice and snow. The text ’This is why I have trust issues’ is a playful way to suggest that these contradictions can lead to distrust or confusion. The humor in this meme is derived from the unexpected contrast between the names of the countries and their actual physical characteristics. | 1.0
```

Listing 4: LLM judge prompt for MMVet

```
Please read the following example. Then extract the answer from the model response and type it at the end of the prompt.

Hint: Please answer the question requiring an integer answer and provide the final value, e.g., 1, 2, 3, at the end.
Question: Which number is missing?
Model response: The number missing in the sequence is 14.
Extracted answer: 14

Hint: Please answer the question requiring a floating-point number with one decimal place and provide the final value, e.g., 1.2, 1.3, 1.4, at the end.
Question: What is the fraction of females facing the camera?
Model response: The fraction of females facing the camera is 0.6,
which means that six out of ten females in the group are facing the camera.
Extracted answer: 0.6

Hint: Please answer the question requiring a floating-point number with two decimal places and provide the final value, e.g., 1.23, 1.34, 1.45, at the end.
Question: How much money does Luca need to buy a sour apple candy and a butter-scotch candy? (Unit: $)
Model response: Luca needs $1.45 to buy a sour apple candy and a butterscotch candy.
```

```
Extracted answer: 1.45

Hint: Please answer the question requiring a Python list as an answer and provide the final list, e.g., [1, 2, 3], [1.2, 1.3, 1.4], at the end.
Question: Between which two years does the line graph saw its maximum peak?
Model response: The line graph saw its maximum peak between 2007 and 2008.
Extracted answer: [2007, 2008]

Hint: Please answer the question and provide the correct option letter, e.g., A, B, C, D, at the end.
Question: What fraction of the shape is blue?
Choices: (A) 3/11 (B) 8/11 (C) 6/11 (D) 3/5
Model response: The correct answer is (B) 8/11.
Extracted answer: B
```

Listing 5: LLM judge prompt for MathVista