LATTE: Learning to Think with Vision Specialists

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://latte-web.github.io
Code: https://github.com/SalesforceAIResearch/LATTE

Abstract

While open-source vision-language models perform well on simple question-answering, they still struggle with complex questions that require both perceptual and reasoning capabilities. We propose LATTE, a family of vision-language models that have LeArned to Think wiTh vision spEcialists. By offloading perception to state-of-the-art vision models, our approach enables vision-language models to focus solely on reasoning over high-quality perceptual information. To train LATTE, we synthesize and filter a large dataset of 273K multi-modal reasoning traces over perceptual outputs of vision specialists. LATTE trained on this data achieves significant 4-5% gains over baselines across 6 benchmarks covering both perception and reasoning abilities. Ablation studies reveal that the effectiveness of multi-modal reasoning traces depends on the data sources, formats, and quality of thoughts.

1 Introduction

The landscape of real-world vision-language tasks is vast, spanning from basic visual question answering (Antol et al., 2015), fine-grained object recognition to complex multi-step geometric reasoning (Hu et al., 2024a). These tasks demand both perception and reasoning. For instance, a user might photograph a gas price panel and ask how much fuel they can afford within a given budget (Figure 1). Solving this requires a model with strong perception—localizing prices via OCR—and multi-step reasoning to compute the answer. While proprietary models like GPT-4o excel due to extensive data and model size scaling, smaller open-source models still struggle (Ma et al., 2024).

To narrow the gap between large proprietary models and smaller open-source counterparts within a reasonable budget, researchers have explored distilling perception and reasoning from larger vision-language models (Shao et al., 2024; Xu et al., 2025) or specialized vision models (Hu et al., 2024b). Despite these efforts, open-source models continue to lag behind.

We argue that the primary reason for this lag is the perception limitations of small vision-language models. While open-source language models have largely caught up with their proprietary counterparts (Lambert et al., 2024; Bi et al., 2024), vision-language models have yet to master heterogeneous vision capabilities. The computer vision community has historically tackled these capabilities separately—e.g., DepthAnything (Yang et al., 2024) for depth estimation and GroundingDINO (Liu et al., 2023d) for object recognition—while unified models still lag behind (Lu et al., 2024a). Similarly, the human brain dedicates distinct regions to categorical recognition (ventral stream) and spatial reasoning (dorsal stream) (Goodale and Milner, 1992), with the reasoning and language-processing frontal and temporal lobes occupying a different volume (Keller et al., 2012). By contrast, vision-language models remain heavily skewed toward language, treating visual encoders as an afterthought (Deitke et al., 2024).

We depart from the *learning to perceive and reason* paradigm to propose a new approach: *learning to reason with vision specialists* (Figure 2). Rather than expecting a small model to master both perception and reasoning, we leverage decades of advancements in computer vision by relying on specialized vision models to provide perceptual information. This allows the vision-language model to focus exclusively on acquiring perceptual information from vision specialists and reasoning over them—enabling it to '*see further by standing on the shoulders of giants*.' Such a paradigm reduces the burden on models to extract low-level perceptual signals, allowing them to concentrate on higher-level reasoning.

To implement this paradigm, we curate high-

*Work done while interning at Salesforce Research

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 11204–11241
November 4-9, 2025

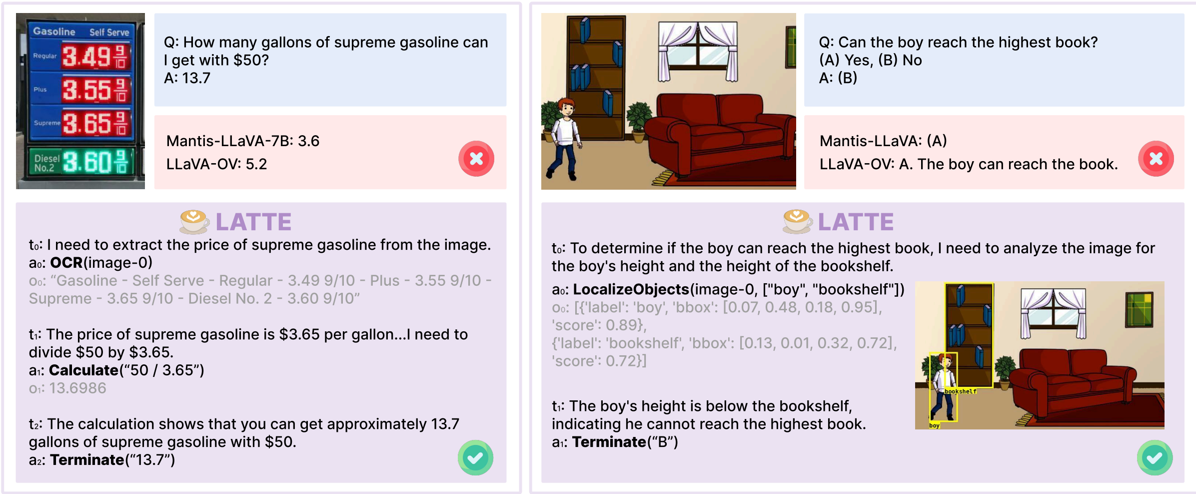


Figure 1: **Example outputs of LATTE vs. SoTA open-source multi-modal language models.** Our LATTE model is able to answer challenging visual questions by reasoning over perceptual information output by vision specialists. It does so by generating a reasoning trace over vision specialists' outputs and producing a final answer.

quality training data in the form of multi-step reasoning traces that integrate perceptual information from vision specialists. We formulate the multi-step reasoning traces as LATTE-trace, where each step consists of: (1) a *thought* for verbalized reasoning; (2) an *action* to retrieve perceptual information from a specific vision specialist; and (3) an *observation* of the returned data. Since obtaining these traces at scale with human annotators is costly, we develop two data engines for synthetic data generation. First, we leverage GPT-4o's strong multimodal reasoning and state-of-the-art vision specialists' precise perception to generate large-scale synthetic reasoning traces across diverse image sources. Second, we generate reasoning traces using Python programs and structured reasoning templates, comparing them against GPT-generated traces to evaluate reasoning quality. In total, we produce over 1M reasoning traces across 31 datasets with GPT-4o and handcrafted programs. We then further apply filtering and mixing techniques and perform extensive experiments with different data ablations.

With the filtered 293K multi-modal reasoning traces, we finetune small 7-8B vision-language models to reason with vision specialists and evaluate our models on 6 benchmarks covering both perception and reasoning skills. We highlight four major takeaways from our experiments: First, learning to reason with vision specialists enables our model to outperform vanilla instruction-tuned baseline by significant margins on both perception and reasoning benchmarks, with an overall average gain of 6.4%. By contrast, the other distillation methods lead to smaller gains or even degradation in the perception performance. Second, our method consistently outperforms the vanilla instruction-tuned baseline by $4-5\%$ on average across all benchmarks regardless of model backbones, with staggering performance gains of $10-20\%$ on MMVet. Third, through data ablations, we confirm that the quality of LATTE-trace matters more than quantity: our best data recipe consists of only 293K LATTE-trace which GPT-4o generated and answered correctly, and it leads to larger performance gains than other data recipes of larger scales. Finally, programmatically-generated LATTE-trace can hurt model performance as a result of the worse reasoning quality, suggesting that again that high-quality reasoning is crucial to the model's performance.

To summarize, we highlight three contributions: (1) We introduce a novel and the largest dataset of 293K multi-modal reasoning traces that cover 31 diverse data sources and include both single- and multi-image questions as well as image-text interleaved traces; (2) We demonstrate the effectiveness of our multi-modal reasoning data and showcase sizable performance gains over baselines on 6 benchmarks through extensive experiments; (3) Finally, our ablation studies reveal new insights into what matters in multi-modal reasoning data. We will release all artifacts publicly.

2 Related work

We contextualize our work on multi-modal language models and multi-modal tool use.

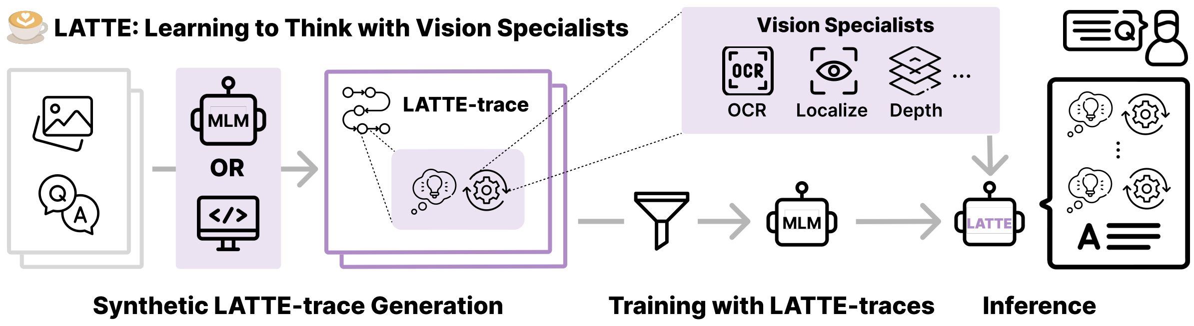


Figure 2: **Overview.** We propose LATTE: learning vision-language models to think with vision specialists via synthetic multi-modal reasoning traces.

Multi-modal language models. Recently, there have been many advances on open-source multi-modal models (Awadalla et al., 2023; Chen et al., 2023; Liu et al., 2023b,a, 2024; Dai et al., 2024; Li et al., 2022, 2023b; Deitke et al., 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., 2024; 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., 2024). 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 learning vision-language models with image-text interleaved reasoning traces.

Multi-modal tool-use. Recently, there is growing interest in training multi-modal language models to be better at tool use (Liu et al., 2023c; Qi et al., 2024; Shao et al., 2024). LLaVa-Plus first shows the possibility of training a multi-modal model to use vision specialists (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., 2024b). Similarly, Visual CoT introduces a new synthetic CoT dataset for training multi-modal models for enhanced reasoning (Shao et al., 2024). More recently, LLaVa-CoT integrates both perception and reasoning from GPT-4o (Xu et al., 2025). Another closely related work CogCoM 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 the authors only experiment with 70K CoM data.

Although these works demonstrate effectiveness, the proposed reasoning datasets are limited in scale and diversity, and none contains multi-image questions or includes images in the reasoning chains (Appendix A Table 9). To complement existing works, we introduce a new large-scale dataset of 293K multi-modal interleaved reasoning traces that cover 31 data sources and include both single-image and multi-image questions.

3 LATTE: Learning to Think with Vision Specialists

Our goal is to train vision-language models to reason about complex multi-modal tasks with the help of vision specialists. To train such models, we need reasoning traces that involve (1) invoking vision specialists and (2) reasoning over their outputs. We refer to such data as LATTE-trace. We define a LATTE-trace $\mathcal{T}$ as a sequence of steps S_i, where each step consists of thought t_i, action a_i and observation o_i:

$$\mathcal{T} = (S_0, S_1, ..., S_n) = (S_i)_{i=0}^{n} \quad (1)$$
$$S_i = (t_i, a_i, o_i), t_i \in L, a_i \in A \quad (2)$$

where L represents language space, and A is the action space consisting of vision specialists. The model only generates t_i and a_i, which the training loss is applied on, whereas o_i is obtained from the vision specialists.

Action space. The action space A of our model consists of vision tools that are either specialized vision models or image processing tools. Concretely, these include OCR (JadedAI, 2025), GETOBJECTS (Zhang et al., 2023), LOCALIZEOBJECTS (Liu et al., 2023d), ESTIMATEOBJECTDEPTH, ESTIMATEREGIONDEPTH (Yang et al., 2024), DETECTFACES (Li et al., 2019), CROP, ZOOMIN,

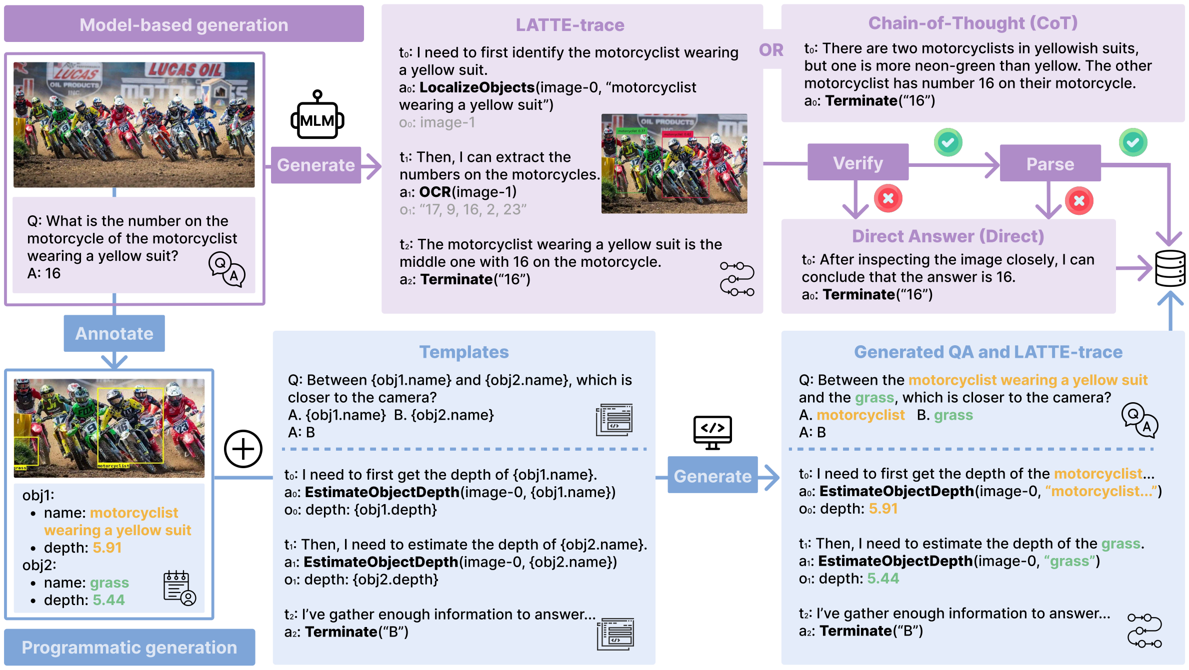


Figure 3: **Data generation.** We illustrate our model-based data generation (top) and programmatic generation (bottom) pipelines.

GetImageToTextsSimilarity, GetImageToImagesSimilarity, GetTextToImagesSimilarity (Radford et al., 2021). Inspired by prior works (Hu et al., 2024a; Gupta and Kembhavi, 2022; Ma et al., 2024), we include a few additional tools to help with reasoning: QueryLanguageModel, QueryKnowledgeBase, Calculate, and SolveMathEquation. We also include Terminate as a tool for the model to output a final answer in the same format. See the Appendix D for all tools' implementation details.

3.1 LATTE-trace generation

We generate synthetic LATTE-trace data with two automatic approaches: Model-based generation and Programmatic data generation.

Model-based generation. This pipeline consists of three steps (Figure 3 top):

1. Generate. First, we leverage images and QA examples in existing visual instruction tuning datasets and generate LATTE-traces to solve the questions with GPT-4o (2024-08-06). 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 following a LATTE-trace or just CoT when it is not necessary (e. g. , the question is straightforward) or not helpful (e. g. , the question requires domain-specific knowledge) to call specialized vision tools (Figure 3). We adopt ReAct-style prompting with JSON-format for calling the vision specialists and provide detailed instructions and examples in the prompt (Yao et al., 2023). All prompts are in the Appendix C.

2. Verify. Second, we verify GPT-4o's generated answers against the ground-truth. We force GPT-4o to always end with Terminate(answer) and compare its prediction to the ground-truth. If the final answer is correct, we move this LATTE-trace to the next stage. Otherwise, we convert this example into the direct answer (Direct) format with the ground-truth (Figure 3).

3. Parse. Finally, we check the JSON syntax of each step of the LATTE-trace. Similar to the previous stage, we again keep the LATTE-traces free of errors and turn the others into the Direct format with ground-truth answers.

Programmatic data generation. In addition to distilling reasoning from proprietary models, we implement a programmatic data generation engine for synthesizing LATTE-traces (Figure 3 bottom) and experiment with these data. This pipeline involves two steps:

1.Annotate. First, we gather existing dense annotations of images. We adopt Visual Genome

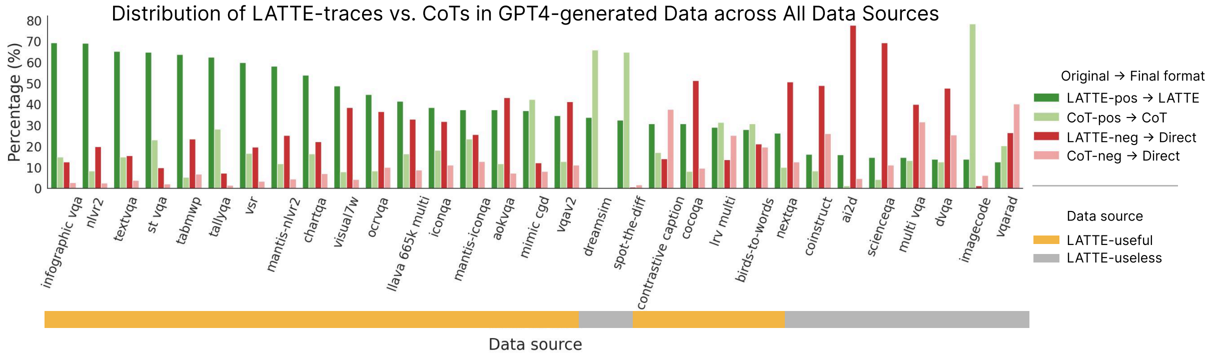


Figure 4: **Distribution of data formats and sources.** We visualize the frequency of data formats (i.e. LATTE-pos/neg, and CoT-pos/neg, pos = correct final answers, neg = incorrect) in the original GPT-4-generated data and in our training data (i.e. LATTE-trace, CoT, or Direct) across all data sources. We also highlight the LATTE-useless (i.e. % of CoT-pos $-$ LATTE-pos $>$ 10 or % of LATTE-neg $-$ LATTE-pos $>$ 10) vs. LATTE-useful datasets.

(VG) as it contains rich human annotations of objects, attributes, and relationships of the images. In addition, we obtain depth maps of the VG images with Depth-Anything-v2 (Yang et al., 2024).

2. GENERATE. Next, we programmatically generate both the QA pairs and the corresponding LATTE-traces with manually written templates and dense image annotations. We reuse the pipeline from (Zhang et al., 2024a,b) to generate QA pairs that cover various vision capabilities such as counting and spatial understanding (See Appendix E.2 for details). To generate LATTE-traces, we define templates for thoughts, actions, and observations across all steps and fill in the templates with the collected annotations. In particular, we manually design five thought templates for each action and randomly sample one during generation. As for actions, we manually select the specialized vision 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.

3.2 Data filtering and mixing

We develop 3 filtering/mixing techniques, where we vary the distribution of: (1) data formats; (2) data sources; and (3) model- vs. program-generated reasoning traces.

Data format. Model-generated data can be categorized into two formats: LATTE-trace or CoT examples (Figure 3). Additionally, they are further grouped into LATTE-trace/CoT-pos and LATTE-trace/CoT-neg examples where the final answers are correct and wrong respectively (Figure 4). Note that we convert both LATTE-trace-neg and CoT-neg examples into the Direct format with ground-truth answers (Figure 3) so the final data format is one of LATTE-trace, CoT, and Direct.

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 vision specialists than others. To this end, we define LATTE-useless datasets as the ones where GPT-4o either decides to output CoT much more often than LATTE-trace (i.e. % of CoT-pos $-$ LATTE-trace-pos $>$ 10), or reaches wrong answers much more frequently than correct ones when using LATTE-trace (i.e. % LATTE-trace-neg $-$ LATTE-trace-pos $>$ 10) (Figure 4). The remaining datasets are considered LATTE-useful datasets.

Program-generated data. As the distribution of actions in model-generated data is imbalanced, with a couple of actions such as GETOBJECTS and OCR dominating the dataset, we also try increasing action diversity by adding programmatic traces with underrepresented actions such as LOCALIZEOBJECTS, and ESTIMATEREGIONDEPTH.

4 Experiments

We perform extensive experiments with small 7-8B multi-modal models and various data recipes on 6 benchmarks to study two questions: (1) do LATTE-traces improve small vision-language models' performance on both perception and reasoning VQAs? (2) what matters in LATTE-traces?

Models. We adopt models with multi-image support as our reasoning traces include multiple images. For most experiments, we use Mantis-8B-SigLIP-LLaMA-3 as the base model. We additionally experiment with Mantis-8B-CLIP-LLaMA-3, and LLaVA-OneVision-7B (Qwen2-7B and SigLIP) to showcase our method's generalizability.

Table 1: **LATTE vs. Vanilla IT with Different Models.** LATTE leads to performance gains over Vanilla IT regardless of the base models. The gains are 4-5% on average across all 6 benchmarks and up to 17% on MMVet.

Language / Vision	Starting checkpoint	Method	**Perception**				**Perception + Reasoning**				**Overall**
			CV-Bench	BLINK	RealWorldQA	Avg	MathVista	MMStar	MMVet	Avg	Avg
LLaMA3-8B / CLIP	Mantis Pretrained	Vanilla IT	52.6	45.8	52.3	50.2	33.1	36.7	28.9	32.9	41.6
		LATTE	56.9	49.6	51.1	52.6	36.6	40.8	45.2	40.8	46.7 (+5.1)
LLaMA3-8B / SigLIP	Mantis Pretrained	Vanilla IT	52.3	43.7	51.8	49.3	31.1	40.5	33.0	34.9	42.1
		LATTE	57.2	47.8	53.7	52.9	34.9	44.6	45.2	41.6	47.2 (+5.1)
	Mantis Instruct-tuned	Vanilla IT	50.6	46.7	54.8	50.7	36.2	40.7	29.7	35.5	43.1
		LATTE	51.7	47.3	56.1	51.7	38.9	45.1	50.0	44.7	48.2 (+5.1)
Qwen2-7B / SigLIP	LLaVa-OV Stage 1.5	Vanilla IT	56.8	50.3	57.8	55.0	42.4	50.1	39.3	43.9	49.5
		LATTE	**60.2**	**52.6**	**61.1**	**58.0**	**46.9**	**50.8**	**50.9**	**51.2**	**53.8** (+4.3)

Table 2: **LATTE vs. Distillation Baselines.** LATTE brings substantial gains over the Vanilla IT baseline on both perception and perception + reasoning benchmarks, whereas VPD and LLaVa-CoT result in smaller gains. LLaVa-CoT even suffers from performance drop in perception tasks. All models were trained with 98K data.

Method	**Perception**				**Perception + Reasoning**				**Overall**
	BLINK	CV-Bench	RealWorldQA	Avg	MathVista	MMStar	MMVet	Avg	Avg
Vanilla IT	44.1	49.2	41.4	44.9	31.0	39.7	27.8	32.8	38.9
VPD	41.6	48.8	**44.8**	45.1 (+0.2)	33.0	41.1	32.8	35.7 (+2.8)	40.4 (+1.5)
LLaVa-CoT	42.2	40.4	38.0	40.2 (-4.7)	36.7	**44.6**	40.2	40.5 (+7.7)	40.4 (+1.5)
LATTE	**46.4**	**54.0**	42.0	**47.5** (+2.6)	**36.9**	44.2	**47.9**	**43.0** (+10.2)	**45.2** (+6.4)

Baselines. We compare LATTE to three types of baselines: (1) vanilla instruction-tuning (IT): instruction-tuning with only direct answers; (2) distillation methods that distill both perception and reasoning from larger models into smaller models, including VPD (Hu et al., 2024b)[1], LLaVa-CoT (Xu et al., 2025), and VisCoT (Shao et al., 2024)[2]; For fair comparison, we train our models and baselines with the same base model, the same hyperparameters, and the same number of examples; (3) multi-modal agents that use tools at inference time, including LLaVa-Plus (Liu et al., 2023c) and CogCoM (Qi et al., 2024).

Training details. We finetune models starting from checkpoints at different stages – pretrained and instruction tuned for Mantis-8B-SigLIP-LLaMA-3, and stage 1.5 for LLaVA-OneVision-7B – to investigate if and where LATTE-traces bring gains. We adopt the hyperparameters from (Liu et al., 2024; 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 additionally perform hyperparameter tuning with LLaVA-OneVision-7B and include this result in the Appendix F.

[1]As VPD is close-sourced, we reproduce their data by converting LATTE-traces into CoTs in VPD's format.

[2]Since VisCoT only has reasoning steps for one data source GQA, training with its data leads to much worse performance. We include its results in the Appendix B.3.

Evaluation setup. We select 6 VQA benchmarks covering both perception and reasoning. The perception-focused benchmarks include RealWorldQA, CV-Bench and BLINK (Tong et al., 2024; Schwenk et al., 2022; Fu et al., 2024; Li et al., 2023a). We also include 3 benchmarks that additionally test reasoning capabilities: MathVista, MMStar, and MMVet (Lu et al., 2024b; Chen et al., 2024a; Yu et al., 2024). We adapt VLMEvalKit (Duan et al., 2024) for our evaluation, where an LLM judge (i.e. GPT-4-turbo) is used to score predictions between 0 and 1 compared to the groundtruth short answers for open-ended questions. Additional details are in Appendix G.

4.1 Do LATTE-traces improve models' performance on both perception and reasoning VQAs?

LATTE beats Vanilla IT on average across all benchmarks regardless of the base model and checkpoint, with significant gains of up to 17% on MMVet. We fine-tune 3 different multi-modal models with all 293K LATTE-traces starting from different checkpoints. We observe that our method leads to consistent gains of 4-5% in the model's average accuracy across 6 benchmarks compared to the baselines instruction-tuned with the same examples in the Direct format (Table 1). We note that our method results in staggering gains of up

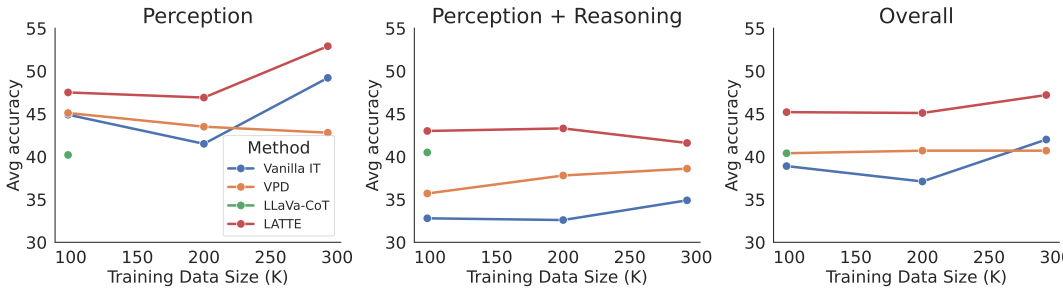


Figure 5: **LATTE vs. Distillation Baselines across Training Data Scales.** LATTE leads to consistent gains on perception and reasoning benchmarks over the Vanilla instruction-tuned baseline across varying training data sizes – 98K, 200K and 293K – and the gains are larger than VPD's. LLaVa-CoT only has 98K data.

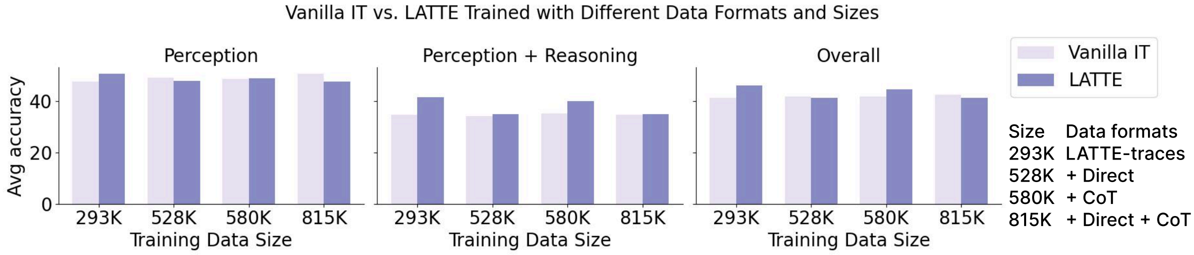


Figure 6: **Ablations on Data Formats.** 293K LATTE-traces lead to the greatest gains over Vanilla IT and the highest overall performance. Adding either CoT or Direct doesn't bring additional gains despite the increased size.

to 17% on MMVet, which covers a wide range of perceptual and reasoning capabilities.

LATTE leads to substantial gains over vanilla instruction-tuning on both perception and reasoning benchmarks, whereas distillation baselines result in smaller gains or even degradation on some perception tasks. We find that learning to reason with vision specialists enables our model to achieve consistent gains on perception-focused VQA benchmarks as well as benchmarks that require both perception and reasoning, with average gains of 2.6% and 10.2% respectively (Table 2). By contrast, both distillation baselines VPD and LLaVa-CoT bring much smaller gains, with an average of 1.5% across all benchmarks, compared to ours (6.4%). Further, we observe that the same trend holds as we scale the training data size from 98K to 200K and 293K, where our method consistently brings larger gains on both perception and perception + reasoning benchmarks (Figure 5). Interestingly, LLaVa-CoT even hurts the model's performance on perception benchmarks, even though it increases the performance on the perception + reasoning benchmarks (Table 2). This result suggests that GPT4-o might still be inferior to vision specialists on some perception tasks, as LLaVa-CoT distills purely from GPT4-o.

LATTE scores higher on MathVista and MMVet than multi-modal agent baselines do, and LATTE-traces bring larger gains to the base model. We see in Table 3 that LATTE achieves higher accuracies on MathVista and MMVet. Moreover, LATTE-traces bring much larger gains to the base model than LLaVa-Plus and CogCoM's data do, despite its comparable size.

Table 3: **LATTE vs. Multi-modal Agent Baselines.** Training with LATTE-traces leads to much larger gains.

Model	Data size	Base model → Finetuned model	
		MathVista	MMVet
LLaVA-Plus	158K	—	32.5 → 35.0 (+2.5)
CogCoM	70K	34.8 → 35.7 (+0.9)	45.9 → 46.1 (+0.2)
LATTE	98K	32.7 → 36.9 (+4.2)	34.4 → 47.9 (+13.5)
LATTE	293K	32.7 → 38.9 (**+6.2**)	34.4 → 50.0 (**+15.6**)

4.2 What matters in LATTE-traces?

We perform ablations with LATTE-traces to study what matters in improving models' performance. For model-generated data, we explore two data filtering techniques on (1) data formats and (2) data sources (Figure 4).

Data quality matters more than quantity: 293K LATTE-traces lead to higher performance than larger mixtures of LATTE-traces and CoT or Direct. We find that 293K LATTE-traces result in the biggest gain of 5% on average over the baseline across all benchmarks (Figure 6). Adding CoT

Table 4: **Ablations on data sources.** Including all sources hurts model's perception and overall performance while having only LATTE-useful datasets helps.

Data source	Size	Method	Percept.	P. + Reason.	Overall
All datasets	815K	Vanilla IT	50.7	34.7	42.7
		LATTE	47.7 (-3.0)	35.1 (+0.4)	41.4 (-1.4)
LATTE-useful datasets	566K	Vanilla IT	46.3	33.3	39.8
		LATTE	46.8 (+0.5)	35.6 (+2.3)	41.2 (+1.4)

examples results in a smaller gain of 2.6%, even though the training data size almost doubles (Figure 6). On the other hand, combining LATTE-trace and Direct examples hurts the model's performance compared to LATTE-traces only, especially on the perception tasks (Figure 6). We empirically observe that models trained with a mix of LATTE-traces and Direct examples tend to adopt the Direct format more often (around 70%) at inference time, relying on its own weaker perceptual ability instead of vision specialists' and thus scoring lower.

Data sources matter too: including all datasets hurts performance while including only LATTE-useful datasets brings gains. Similarly, we see that including only the LATTE-useful datasets – where GPT-4o frequently chooses to use vision specialists and reaches correct final answers – improves the model's average performance compared to the baseline, while including all data sources doesn't (Table 4). Again, we see that a smaller set of 566K LATTE-traces leads to better performance than a much larger dataset (815K), implying that data quality matters more than quantity.

Table 5: **Ablations on programmatic LATTE-traces.** We find that training with additional programmatic LATTE-traces doesn't bring more gains.

M: P	Data format	Size	MathVista	Percept. + Reason.	Overall
—	Direct		31.1	34.9	42.0
0:1	P-traces	293K	17.3	15.9	27.2
1:0	M-traces		34.9	<u>41.6</u>	**47.2**
1:0.1	+P-traces 29K	322K	33.9	40.1	44.0
1:0.25	+P-traces 73K	366K	**38.3**	**42.1**	<u>46.3</u>
1:0.5	+P-traces 147K	440K	<u>36.7</u>	39.7	45.5
1:1	+P-traces 293K	586K	31.0	36.2	43.2

Programmatically generated LATTE-traces can help on a certain benchmark but not overall, likely due to the worse quality of thoughts. We experiment with a mixture of model-generated and programmatic reasoning traces, with ratios ranging from 1:0.1 to 1:1. We find that training with only programmatic LATTE-traces results in large performance drops (Table 5). Similarly, while adding programmatic LATTE-traces can bring gains on some benchmark (e.g. MathVista), it fails to bring overall gains despite the increased data size (Table 5). This is likely due to the model's worse reasoning capability learned from templated thoughts. See more details in Appendix B.2 (Figure 8).

Overall, our experiments suggest that the quality of perceptual information and reasoning are both crucial to improving vision-language models' performance across diverse VQAs.

4.3 Additional ablations

Table 6: **Ablations on LATTE's inference setup.** The OCR tool greatly affects model's performance, while the query LLM tool doesn't; and increasing the maximum number of tool calls doesn't help beyond 10.

Method	Percept.	P. + Reason.	Overall
LATTE (max 10 calls)	51.7	43.8	**47.8**
max 5 calls	51.7	42.8	47.2
max 20 calls	51.6	43.6	47.6
no QUERYLM	52.1	43.5	**47.8**
OCR with easyocr	51.4	39.9	45.7

What matters in LATTE's inference setup? In addition to training data, we also perform ablations on LATTE's inference setups, including tool ablations and tool call number ablations.

First, the tool ablations include (1) excluding the QUERYLANGUAGEMODEL tool, and (2) using EasyOCR (JadedAI, 2025) instead of GPT for OCR. We see that removing QUERYLANGUAGEMODEL leads to no difference in the model's overall performance (Table 6), suggesting that QUERYLANGUAGEMODEL is not an important tool to the VQA benchmarks in our evaluation. Nevertheless, by using EasyOCR for OCR we do see a large drop in the model's performance, especially on reasoning-intensive benchmarks such as MMVet due to EasyOCR's less accurate outputs (Table 6). Still, LATTE is over 2% above the vanilla instruction-tuned baseline on average across all benchmarks.

Second, as we adopt multi-turn conversations for all models with our method, we can vary the number of tool calls at inference time. We set the maximum number of tool calls to 10 by default and experiment with 5 and 20 tool calls. Through this experiment, we see that the model's scores peak at 10 maximum tool calls, suggesting that 10 tool calls are sufficient in most cases (Table 6).

Do LATTE-traces improve stronger base models? We further ablate on the model backbone, as new and stronger open-source vision-language models come out. Our results using the latest

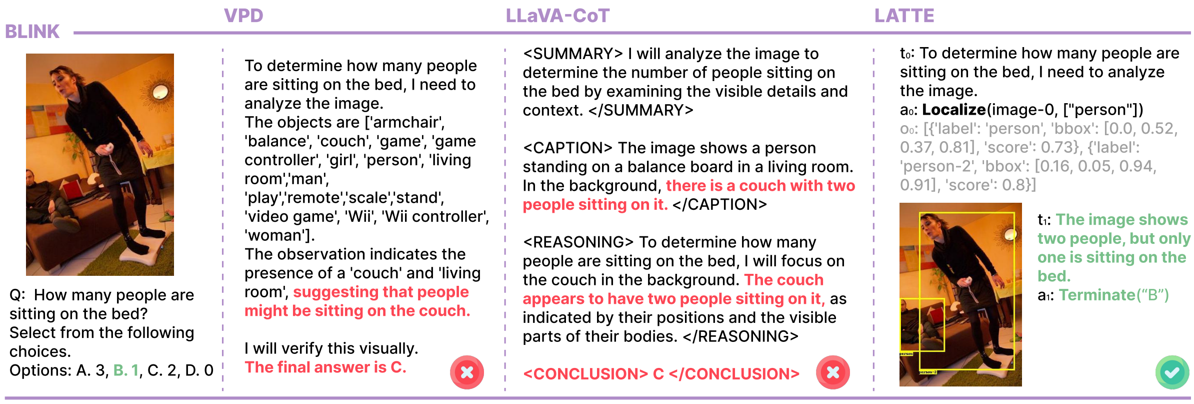


Figure 7: **Qualitative analysis.** Example outputs of VPD, LLaVA-CoT vs. LATTE on BLINK.

Table 7: **Ablations on LATTE's model backbone.** LATTE-traces improve stronger base models too.

Model backbone	Method	Percept.	P. + Reason.	Overall
InternVL3-8B	Vanilla IT	51.5	45.1	46.7
	LATTE	54.1	54.5	**52.0**
Qwen2.5VL-3B	Vanilla IT	54.3	49.6	50.8
	LATTE	53.4	55.7	**55.1**
Qwen2.5VL-7B	Vanilla IT	53.7	52.5	52.8
	LATTE	56.9	57.7	**57.5**

vision-language models – Qwen2.5VL and InternVL3 – as the base models demonstrate that our method improves upon vanilla instruction tuning even with strong base models (Table 7).

4.4 Error Analysis

Where does LATTE perform better than the distillation baselines? We find that LATTE performs better in fine-grained perception tasks such as the counting questions in BLINK, while VPD and LLaVA-CoT tend to hallucinate and make perceptual errors (Figure 7).

Table 8: **LATTE's Error Types on MMVet.**

Error type	Subtype	%
Tool call	Format	3
	Value	6
Tool result	—	7
Model perception	Not using tools	6
	Irrelevant tools	50
Model reasoning	—	28

What errors does LATTE make? On MMVet, we find that the model's most frequent error happens when it falls back on its own perceptual ability after deciding not to use tools or finding the vision tools' outputs irrelevant/not helpful for the question (e. g. movie, arts, or medical questions that require domain knowledge) (Table 8). These numbers suggest that the model's performance can be improved by diversifying tools and questions in the training data, and strengthening reasoning.

5 Conclusion

We propose to learn vision-language models to reason with vision specialists. To learn such models, we synthesize a novel large-scale dataset of multimodal reasoning traces grounded on perceptual information. With this data, we fine-tune small vision-language models and perform extensive experiments. Across 6 benchmarks covering both perception and reasoning, we demonstrate that our model achieves significant gains over vanilla instruction-tuned baselines and other distillation methods in perception and reasoning tasks.

6 Limitations

First, our method requires customized implementations of the specialized vision tools. Second, reasoning with the vision specialists requires additional compute at inference time. Nevertheless, it is becoming a common practice to increase model performance by scaling up test time compute (OpenAI, 2025; Muennighoff et al., 2025). Future work can optimize and enhance the implementations of vision specialists, especially as the computer vision community continues to advance vision models. Additionally, while we try to include the most important tools for general perception and reasoning, other types of VQA e. g. knowledge-intensive ones might benefit from additional tools as suggested in our error analysis. Lastly, due to the limited generalization of supervised finetuning and diversity of the visual world, researchers might need to explore training alteratives (e. g. reinforcement learning) for better generalization or train new models with different vision specialists for other applications (e. g. web navigation) or for other domains (e. g. medical question answering).

7 Acknowledgement

Zixian Ma was partially funded by Sony for this project. Zixian Ma conducted most of this work at a Salesforce internship.

References

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

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. 2023. Openflamingo: An open-source framework for training large autoregressive vision-language models. *Preprint*, arXiv:2308.01390.

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, and 1 others. 2024. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*.

Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, and 1 others. 2024a. Are we on the right way for evaluating large vision-language models? *arXiv preprint arXiv:2403.20330*.

Qiguang Chen, Libo Qin, Jin Zhang, Zhi Chen, Xiao Xu, and Wanxiang Che. 2024b. M ^{3} cot: A novel benchmark for multi-domain multi-step multi-modal chain-of-thought. *arXiv preprint arXiv:2405.16473*.

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. 2023. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. *arXiv preprint arXiv:2312.14238*.

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

Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, and 1 others. 2024. Molmo and pixmo: Open weights and open data for state-of-the-art multimodal models. *arXiv preprint arXiv:2409.17146*.

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. 2024. Vlmevalkit: An open-source toolkit for evaluating large multi-modality models. *Preprint*, arXiv: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. 2024. Blink: Multimodal large language models can see but not perceive. *arXiv preprint arXiv:2404.12390*.

Melvyn A Goodale and A David Milner. 1992. Separate visual pathways for perception and action. *Trends in neurosciences*, 15(1):20–25.

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

Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, and Ranjay Krishna. 2024a. Visual sketchpad: Sketching as a visual chain of thought for multimodal language models. *Preprint*, arXiv:2406.09403.

Yushi Hu, Otilia Stretcu, Chun-Ta Lu, Krishnamurthy Viswanathan, Kenji Hata, Enming Luo, Ranjay Krishna, and Ariel Fuxman. 2024b. 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*, pages 9590–9601.

JadedAI. 2025. Easyocr.

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

Georg B Keller, Tobias Bonhoeffer, and Mark Hübener. 2012. Sensorimotor mismatch signals in primary visual cortex of the behaving mouse. *Neuron*, 74(5):809–815.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, and 1 others. 2024. T\" ulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*.

Hugo Laurençon, Léo Tronchon, Matthieu Cord, and Victor Sanh. 2024. What matters when building vision-language models? *Preprint*, arXiv:2405.02246.

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

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

Jian Li, Yabiao Wang, Changan Wang, Ying Tai, Jianjun Qian, Jian Yang, Chengjie Wang, Jilin Li, and Feiyue Huang. 2019. Dsfd: Dual shot face detector. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

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

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

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

Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. 2024. Llava-next: Improved reasoning, ocr, and world knowledge.

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

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. 2023c. Llava-plus: Learning to use tools for creating multimodal agents. *Preprint*, arXiv:2311.05437.

Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, and 1 others. 2023d. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*.

Jiasen Lu, Christopher Clark, Sangho Lee, Zichen Zhang, Savya Khosla, Ryan Marten, Derek Hoiem, and Aniruddha Kembhavi. 2024a. Unified-io 2: Scaling autoregressive multimodal models with vision language audio and action. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 26439–26455.

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

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

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. *Preprint*, arXiv:2501.19393.

OpenAI. 2025. Openai o3-mini.

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

Alec Radford, Jong Wook Kim, Chris Hallacy, A. Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning transferable visual models from natural language supervision. In *ICML*.

Tanik Saikh, Tirthankar Ghosal, Amish Mittal, Asif Ekbal, and Pushpak Bhattacharyya. 2022. Scienceqa: A novel resource for question answering on scholarly articles. *International Journal on Digital Libraries*, 23(3):289–301.

Dustin Schwenk, Apoorv Khandelwal, Christopher Clark, Kenneth Marino, and Roozbeh Mottaghi. 2022. A-okvqa: A benchmark for visual question answering using world knowledge. *Preprint*, arXiv:2206.01718.

Hao Shao, Shengju Qian, Han Xiao, Guanglu Song, Zhuofan Zong, Letian Wang, Yu Liu, and Hongsheng Li. 2024. Visual cot: Unleashing chain-of-thought reasoning in multi-modal language models. *Preprint*, arXiv:2403.16999.

Shengbang Tong, Zhuang Liu, Yuexiang Zhai, Yi Ma, Yann LeCun, and Saining Xie. 2024. Eyes wide shut? exploring the visual shortcomings of multimodal llms. *Preprint*, arXiv:2401.06209.

Penghao Wu and Saining Xie. 2024. V?: Guided visual search as a core mechanism in multimodal llms. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13084–13094.

Guowei Xu, Peng Jin, Hao Li, Yibing Song, Lichao Sun, and Li Yuan. 2025. Llava-cot: Let vision language models reason step-by-step. *Preprint*, arXiv:2411.10440.

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

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. *Preprint*, arXiv:2210.03629.

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

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

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. 2024b. Provision: Programmatically scaling vision-centric instruction data for multimodal language models. *Preprint*.

Youcai Zhang, Xinyu Huang, Jinyu Ma, Zhaoyang Li, Zhaochuan Luo, Yanchun Xie, Yuzhuo Qin, Tong Luo, Yaqian Li, Shilong Liu, and 1 others. 2023. Recognize anything: A strong image tagging model. *arXiv preprint arXiv:2306.03514*.

A Dataset and model comparison

We summarize the differences between our work and the other multi-modal CoT datasets including ScienceQA, M^3COT, Visual CoT, VPD, V*, LLaVa-Plus, LLaVA-CoT, CogCoM in Table 9.

B Additional results

B.1 Additional qualitative examples

We present additional successful outputs of LATTE across both single-image and multi-image examples in Figures 10 and 11 as well as failure cases in Figure 12.

B.2 Qualitative error analysis

Why does adding programmatic LATTE-trace help on MathVista but hurt MMVet performance? We observe that adding programmatic LATTE-trace can result in up to 3% gain on MathVista and 9% drop on MMVet. Upon analysis, we discover that programmatic LATTE-trace 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 LATTE to learn to use it effectively (Figure 8). Conversely, programmatic data hurts LATTE'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 8).

B.3 Additional quantitative results

We report additional quantitative results of data ablations on Mantis-CLIP in Table 14, where we see the same trends we observe with Mantis-SigLIP: the smallest dataset of 293K LATTE-trace examples leads to the highest absolute performance and gain compared to other datasets with a mix of LATTE-trace, CoT, and/or Direct examples at larger scales.

Visual-CoT Performance. We experimentally compare LATTE to Visual-CoT. We finetune Mantis-LLaVA-Pretrained (LLama3+SigLIP) with Visual CoT and compare its performance with LATTE (Table 10). We use 413K examples where the bounding boxes are valid and within the image. We find that the models trained with Visual CoT data achieve an average accuracy of 39.3% (much lower as Visual COT's data are mostly Text/Doc images and contain only bboxes without natural language thoughts) on the benchmarks.

Performance gain with LATTE inference. We compare the model's performance when trained with a random mix of 293K LATTE-traces and Direct data (1:1) and tested with LATTE format vs. Direct prompt. We find that the model achieves an average of 50.3% when tested following LATTE format vs. 48% with the Direct prompt (Table 11), suggesting that reasoning with vision specialists at inference time improves model's performance.

Hyperparameter tuning 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 9).

C Model-based data generation

C.1 Generation prompt

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

C.2 Dataset statistics

We present a table with detailed statistics of the LATTE-trace 293K dataset in Table 15.

D Action implementation

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

E Programmatic data generation

E.1 QA and action templates

We present the question-answer and corresponding action templates used in our programatic data generation in Table 16. 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.

E.2 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.

Table 9: **Dataset and model comparison.**

Paper	Dataset					Model	
	Training set size	Data source number	Tool number	Multi-image questions	Multi-modal reasoning chain*	Inference-time tool-use	Multi-image support
Science QA (Saikh et al., 2022)	12.6K	1	✗	✗	✗	✗	✗
M3CoT (Chen et al., 2024b)	7.8K	2	✗	✗	✗	✗	✗
VPD (Hu et al., 2024b)	90K	6	6	✗	✗	✗	✗
LLaVA-CoT (Xu et al., 2025)	100K	10	✗	✗	✗	✗	✗
V* (Wu and Xie, 2024)	206K	3	1	✗	✗	✗	✓
LLaVA-Plus (Liu et al., 2023c)	158K	6	12 (or 19 counting compositional ones)	✗	✗	✓	✗
VisualCoT (Shao et al., 2024)	98K (+340K with bboxes but no thoughts)	12	2	✗	✗	✓	✓
CogCoM (Qi et al., 2024)	70K	3	6	✗	✗	✓	✓
LATTE	293K (+over 1M program generated reasoning traces)	31	15 tools (see Section 2)	✓	✓	✓	✓

*The reasoning chain contains not just texts but also images.

Table 10: **VisCoT results**

Method	A-OKVQA	BLINK	MathVista	MMMU	MMStar	MMVet	MMVP	RealWorldQA	Avg
VPD-LLaVA	80.7	43.9	35.1	40.8	40.1	40.6	61.3	45.1	48.5
VisCoT-LLaVA	67.9	39.4	12.9	36.1	36.0	26.7	61.3	34.0	39.3

Table 11: **LATTE vs. Direct inference**

Test prompt	A-OKVQA	BLINK	MathVista	MMMU	MMStar	MMVet	MMVP	RealWorldQA	Avg
LATTE	82.1	45.9	37.0	39.8	43.4	46.7	66.0	41.6	50.3
Direct	81.9	46.6	31.2	42.1	39.1	35.0	68.3	40.2	48.0

E.3 Example action distribution

We plot example distributions of all actions before and after adding programmatic LATTE-trace 73K data in Figure 13.

F Additional training details

We report additional training hyperparameters for Mantis models and LLaVA-OV in Table 12 and 13 respectively.

Table 12: **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

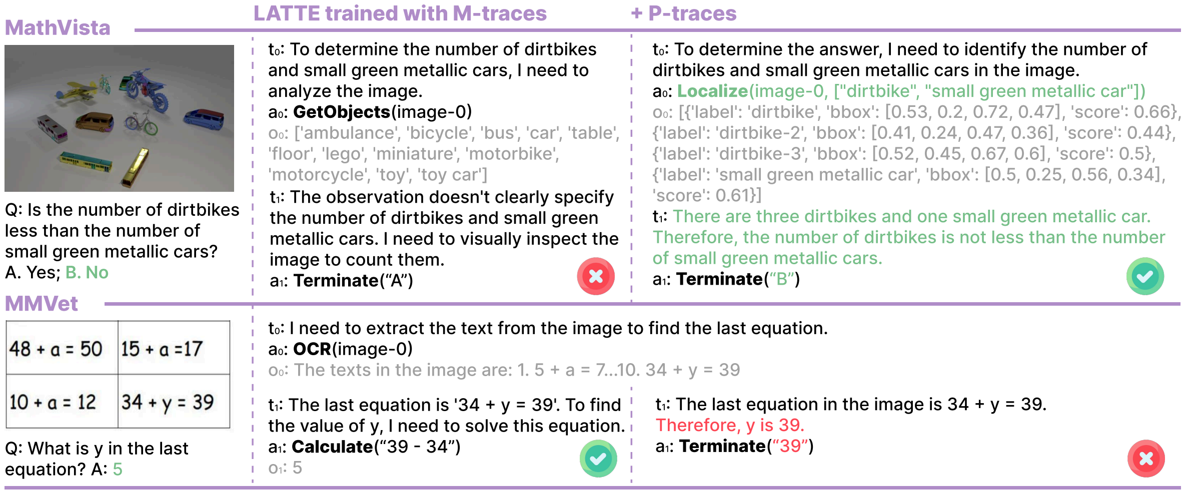


Figure 8: **Qualitative analysis.** Examples of LATTE success and failure after adding programmatic LATTE-traces.

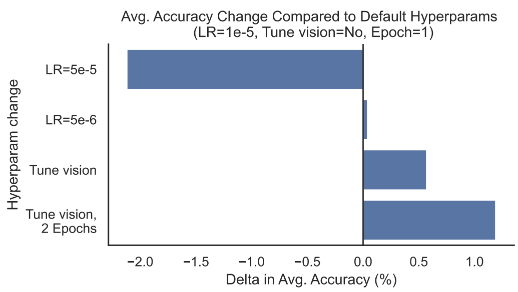


Figure 9: **Hyperparameter ablations.** Additional gains can be achieved with a smaller learning rate for the language model, tuning the vision encoder, and training for more epochs.

Table 13: **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

G Additional evaluation details

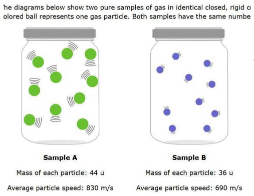
We present additional inference and evaluation details in Table 17 and the LLM judge prompts used for MMVet and MathVista from VLMEvalKit (Duan et al., 2024) in Listings 4 and 5.

H License

The datasets and models are for research purposes only in support of an academic paper. All artifacts from this paper are licensed under the noncommercial license CC-BY-NC 4.0. Some of our models were built with Meta Llama 3, which is licensed under the Meta Llama 3 Community License, Copyright © Meta Platforms, Inc. All Rights Reserved.

Table 14: **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	
	LATTE-trace		LATTE	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	
	LATTE-trace+ CoT		LATTE	**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	
	LATTE-trace+ CoT+ Direct		LATTE	81.3	43.1	32.7	**48.0**	37.7	31.0	58.7	47.6	47.5	-1.2
LATTE-useful datasets	Direct	566K	Mantis-CLIP	79.1	44.1	30.9	39.4	35.5	30.1	62.7	49.8	46.5	
	LATTE-trace+ CoT+ Direct		LATTE	80.7	41.6	31.5	41.0	38.2	36.1	59.0	48.2	47.0	0.6



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

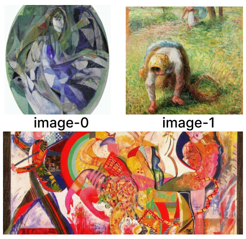
t_0: To determine which sample has a higher temperature, I need to compare the average kinetic energies of the particles in each sample using the formula: KE = 0.5 * m * v^2, where m is the mass and v is the velocity. Since both samples have the same number of particles, the sample with the higher average 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")

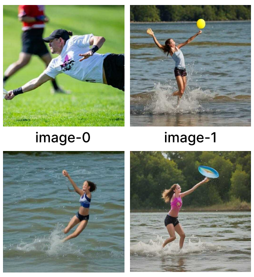
Q: How to cook this dish?
A: This dish is called "Strawberry Rhubarb Pie". This strawberry rhubarb pie is a favorite springtime dessert in our house...Ingredients: 1 cup white sugar, ½ cup all-purpose flour...2 tablespoons white sugar...Step 1: Preheat oven to 400 degrees F (200 degrees C)...Step 8: 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 10: We present additional successful examples of LATTE.

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

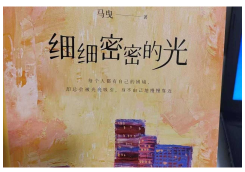
t_0: I need to extract the prices of the margherita pizza and calzone from the image to calculate the total cost.
a_0: **OCR**(image-0)
o_0: The text on the blackboard reads:\n\n**PIZZA**\n\n- Margherita 5.90€\n- Capricciosa 8.50€...Funghi 6.90€\n\n**Calzone 14,00**\n\n**WI-FI FREE inside**'
t_1: The 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: 马叟<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 11: We present additional successful examples of LATTE.

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")

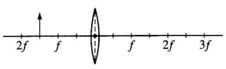
Q: Is the wind blowing the flag?
A. Yes, B. No

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")

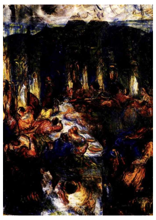
Q: Which type of eggs are in the corner of the picture?
A. Quail eggs, B. Chicken eggs

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

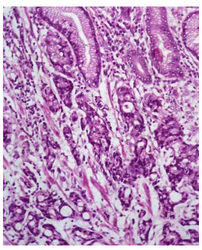


t_0:To determine the type and size of the image formed, we need to analyze the position of the object relative to the focal length.", "actions": [{"name": "", "arguments": {"query": ""}}]}
a_0: **QueryLanguageModel**("What type of image is formed when an object is placed at a distance of 1.5f from a 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 12: We present additional failure cases of LATTE.

Table 15: **Detailed statistics of the LATTE-trace 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

Table 16: **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 Multi-image understanding, Counting Multi-image understanding, Counting Multi-image understanding, Counting Multi-image understanding, Attribute recognition Multi-image understanding, Attribute recognition, Counting	Which image has {object}? How many {object} are in in these images? Which image has most {object}? Which image has least {object}? Which image has {attribute} {object}? How many {attribute} {object} in these images?	LocalizeObjects x N

Table 17: **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

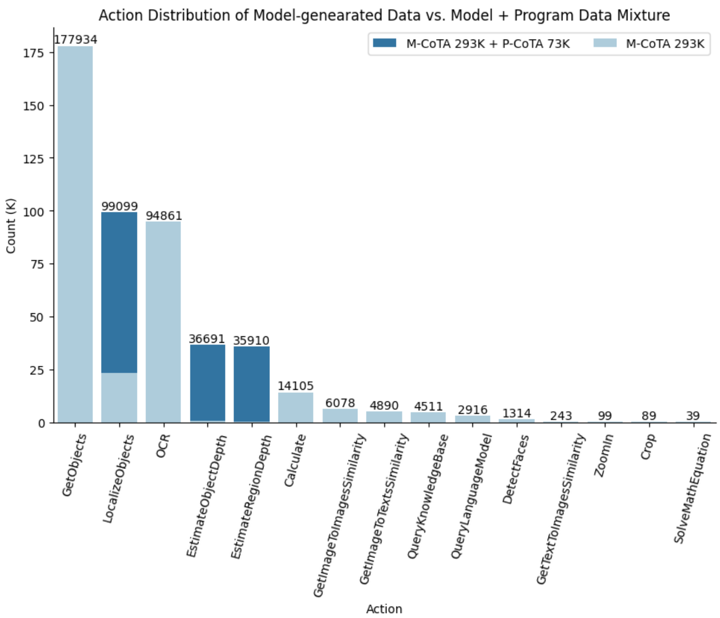


Figure 13: Action distribution of model-generated data vs. model and program data mixtures.

```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 = {}
        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:
```

```python
            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}


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.",
```

```python
            "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.",
```

```python
                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',
            ))
```

```python
        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

```python
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