# Visual SKETCHPAD: Sketching as a Visual Chain of Thought for Multimodal Language Models

**Anonymous Author(s)**

## Abstract

Humans draw to facilitate reasoning: we draw auxiliary lines when solving geometry problems; we mark and circle when reasoning on maps; we use sketches to amplify our ideas and relieve our limited-capacity working memory. However, such actions are missing in current multimodal language models (LMs). Current chain-of-thought and tool-use paradigms only use text as intermediate reasoning steps. In this work, we introduce SKETCHPAD, a framework that gives multimodal LMs a visual sketchpad and tools to draw on the sketchpad. The LM conducts planning and reasoning according to the visual artifacts it has drawn. Different from prior work, which uses text-to-image models to enable LMs to draw, SKETCHPAD enables LMs to draw with lines, boxes, marks, etc., which is closer to human sketching and better facilitates reasoning. SKETCHPAD can also use specialist vision models during the sketching process (e.g., draw bounding boxes with object detection models, draw masks with segmentation models), to further enhance visual perception and reasoning. We experiment on a wide range of math tasks (including geometry, functions, graph, chess) and complex visual reasoning tasks. SKETCHPAD substantially improves performance on all tasks over strong base models with no sketching, yielding an average gain of 12.7% on math tasks, and 8.6% on vision tasks. GPT-4o with SKETCHPAD sets a new state of the art on all tasks, including $V^*$Bench (80.3%), BLINK spatial reasoning (83.9%), and visual correspondence (80.8%). We will release all code and data.

## 1 Introduction

Sketching is a fundamental human activity, serving as a versatile tool for communication [11], ideation [44], and problem-solving [43]. Unlike written language, sketches have the advantage of conveying visuo-spatial ideas directly, for example by using spatial relations on paper to convey spatial relations or other more abstract relationships in the world. Sketches are so fundamental that we use them to teach school children how to solve geometry problems by drawing support lines, to aid engineers conveying prototypes, to support architects creating blueprints, and to allow scientists like us to convey scientific contributions (see Figure 1).

As multimodal language models (LMs) [34, 38, 27, 26, 2, 3, 45, 7, 30, 37, 6, 5] have begun to mature, we now expect them to solve tasks like the ones mentioned above, i.e., ones where people draw intermediate sketches to simplify reasoning. In recent benchmarks on complex geometry and math problems (e.g., Geometry3K [31], IsoBench [8]), models are given images of diagrams and asked questions requiring symbolic grounding and spatial understanding, where intermediate sketches like auxiliary lines can enhance reasoning. Even benchmarks in computer vision (e.g., BLINK [9], $V^*$Bench [47])now have a similar flavor. Specialist vision models can be viewed as sketching on natural images. For example, object detection is plotting bounding boxes around objects; depth estimation is drawing colormaps according to depth. Unfortunately, current LMs lack a scaffold for using sketch-based reasoning when solving tasks.

In this paper, **we introduce Visual SKETCHPAD: a framework that provides multimodal LMs with the tools necessary to generate intermediate sketches to reason over tasks.** Inspired by textual chain-of-thought reasoning in LMs [46, 57], SKETCHPAD prompts the underlying visual LM

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.

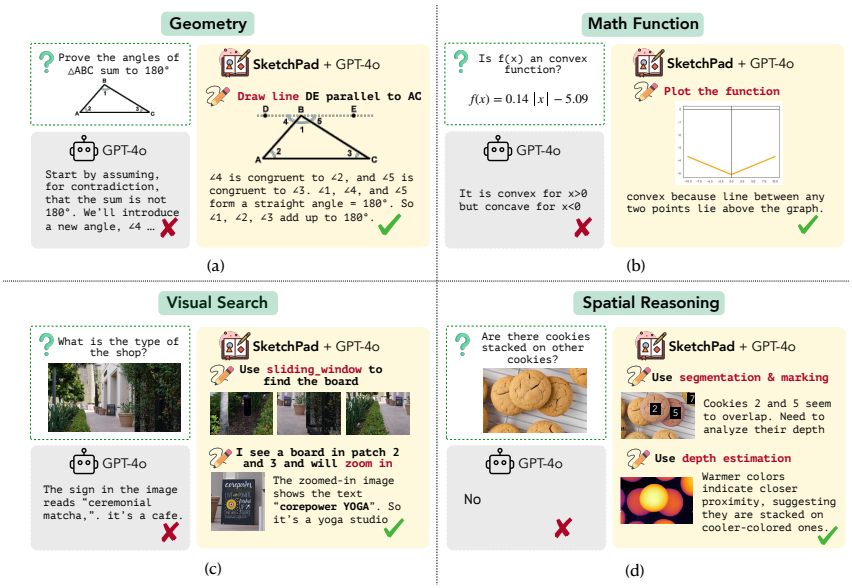


Figure 1: **SKETCHPAD equips GPT-4 with the ability to generate intermediate sketches to reason over tasks.** Given a visual input and query, such as proving the angles of a triangle equal 180°, SKETCHPAD enables the model to draw auxiliary lines which help solve the geometry problem.

to produce visual artifacts as part of a chain of mixed textual, programmatic, and visual reasoning. For example, to prove that the angles of triangles sum up to 180 degrees in Figure 1 (a), SKETCHPAD enables agents to modify the diagram by introducing a new auxiliary line. This new line, along with new annotated angles, provides the critical information to solve the geometry task. Similarly, SKETCHPAD improves models' spatial reasoning for computer vision. To determine if there are cookies stacked on top of other cookies in the image (Figure 1b), the model first produces an intermediate depth estimate. By analyzing the depth estimate, which reveals cookies overlapping at different depths, the model is able to correctly answer that the cookies are indeed stacked.

We demonstrate the effectiveness of visual SKETCHPAD across a wide range of mathematics and computer vision tasks. For math, we tackle problems including (1) geometry [31], (2) mathematical functions, (3) graph algorithms, and (4) strategy games [8]. **Across all four categories of mathematical tasks, SKETCHPAD consistently improves the baseline GPT-4o performance, yielding an average gain of 12.7%.** For computer vision, we tackle diverse tasks including (1) depth, (2) spatial reasoning, (3) jigwaw, (4) visual correspondence, (5) semantic correspondence, as well as questions from (6) the MMVP and (7) the *V**Bench benchmarks [9, 40, 47]. For this domain, SKETCHPAD enables models to generate segmentation masks, crop images, draw bounding boxes, zoom into image regions, overlay images, etc. Similar to math, **SKETCHPAD shows consistent improvements across all seven types of computer vision tasks**. For example, GPT-4o, augmented with SKETCHPAD, sees 14.3% improvement on *V**Bench, 12.1%, and 9.7% improvements on BLINK's depth and semantic correspondence tasks, setting a new state of the arts across all tasks.We hope SKETCHPAD opens up new research opportunities toward more capable and interpretable multimodal intelligence.

## 2 Visual SKETCHPAD

Figure 2 shows examples of how SKETCHPAD works. Our framework requires no finetuning or training. Multimodal LMs, out of the box, can be prompted to sketch using our framework. The SKETCHPAD model solves tasks by iteratively interacting with an environment. Given a multimodal query $q$ consisting of both visual and textual components, the model generates a sequence of thoughts, actions, and receievs observations to gather the necessary information toanswer the query. SKETCHPAD follows a three-step process at each time step $t$:

**Thought:** Given the current multimodal context $c_t$, the model reasons over $c_t$, generating text to support future reasoning. For example, for a geometry problem, the LM may decide to draw an auxiliary line between two points. For a computer vision problem, the LM may decide to draw a bounding box around an object and mark the box with a number.

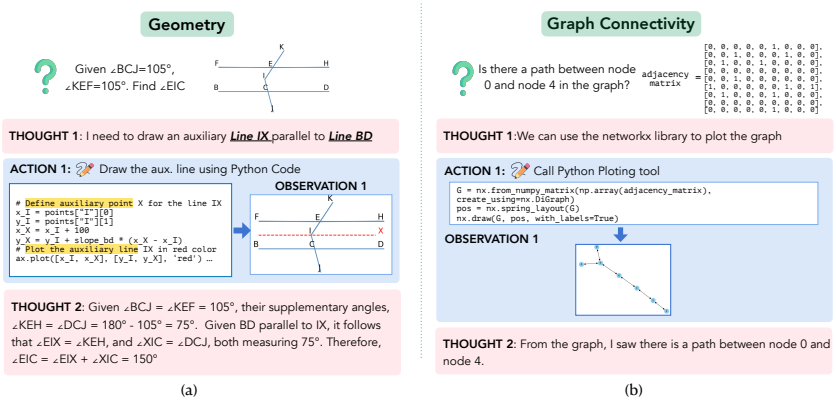


Figure 2: **Overview of SKETCHPAD**. Given a multimodal query, the SKETCHPAD agent generates a sketching plan to address the query (*Thought*), and then synthesizes a program to create visual sketches (*Action*). By analyzing the resulting sketches (*Observation*), which serve as a visual representation of the reasoning process, the model generates a final response to the query.

**Action:** Based on the thought, the model executes action $a_t$. In SKETCHPAD, this action can manipulate multimodal content. For example, the model can write Python codes to realize the proposed Thought. The action is executed; in this case, the generated code will be compiles and executed. We describe this step in detail in §

**Observation:** Based on the action $a_t$, SKETCHPAD's environment returns a new observation $o_{t+1}$. The context is updated to $c_{t+1} = (c_t, a_t, o_{t+1})$. The model iterates with the thought, action, observation steps for time step $t + 1$, reasoning over its prior sketches. The LLM can also choose to return a final answer and terminate the reasoning process.

## 3 Experimental Results

**Experimental setups on math tasks.** we experiment with SKETCHPAD on four complex mathematical tasks : (1) geometry, (2) mathematical functions, (3) graph algorithms, and (4) game strategies. Details of our evaluation tasks and the tools employed for visual reasoning are as in §C. We evaluate the performance of SKETCHPAD on multimodal LMs with API access, including `gpt-4-turbo-2024-04-29` and `gpt-4o-2024-05-13`.

**Main results on math tasks.** As shown in Table 1, SKETCHPAD consistently improves base model performance across all tasks, with an average improvement of 18.8% for `GPT-4o` and 13.5% for `GPT-4 Turbo`. In particular, we observe large gains on graph algorithms such as maximum flow and connectivity. For instance, `GPT-4o` with SKETCHPAD achieves an accuracy of 66.3% on the maximum flow problem, improving over the base model by 41.3%. Similarly, SKETCHPAD substantially improves the performance on mathematical functions, with `GPT-4 Turbo` achieving over 90% accuracy and `GPT-4o` over 88% accuracy on convexity and parity classification tasks. Furthermore, we observe notable gains ($\sim$ 20%) on game strategies. Overall, these results highlight the effectiveness of SKETCHPAD in enhancing the reasoning capabilities of multimodal language models across diverse domains.

**Experimental setups on computer vision tasks.** We experiment with SKETCHPAD on complex visual reasoning tasks. Recent work (BLINK) [9] finds that many core visual perception abilities are still missing from existing multimodal LMs—even though many computer vision specialist models possess such abilities. Also, SoM [51] shows that drawing segmentation masks on images unleashes the strong visual grounding ability of GPT-4V. We generalize these ideas with SKETCHPAD, allowing LMs to use **specialist vision models** to sketch. Details of these modules are in §D.1. Details of the tasks and vision specialists we used are in §D.

**Computer vision tasks results.** Table 2 shows the performance of our SKETCHPAD and baselines. SKETCHPAD consistently improves base model performance across all tasks. `GPT-4o` with SKETCHPAD sets the new state-of-the-art results on all tasks. SKETCHPAD is particularly effective on $V^*$Bench, yielding 18.5% accuracy improvement for `GPT-4 Turbo` and 14.3% improvement for `GPT-4o`, surpassing the previous state of the art SEAL [47] which used a visual search model

| | Geometry | Graph | | | Math | | Game |
|---|---|---|---|---|---|---|---|
| Model | Geometry | Maxflow | Isomorphism | Connectivity | Convexity | Parity | Winner ID |
| *Prior LLMs without visual inputs* | | | | | | | |
| Gemini-Pro | \ | 15.6 | 47.7 | 50.0 | 87.9 | 48.2 | 8.1 |
| Claude 3 OPUS | \ | 56.3 | 50.0 | 82.0 | 93.0 | 77.6 | 74.4 |
| Mixtral 8x7B [18] | \ | 8.6 | 50.0 | 62.5 | 69.1 | 41.7 | 7.4 |
| LLaMA-2-70B [41] | \ | 18.0 | 50.0 | 50.0 | 74.2 | 33.3 | 12.4 |
| *Latest multimodal LLMs + Visual Sketchpad* | | | | | | | |
| GPT-4 Turbo | 37.5 | 32.8 | 62.5 | 66.0 | 57.0 | 80.5 | 50.4 |
| + Sketchpad | 45.8 | 63.3 | 64.2 | 95.1 | **93.1** | **93.1** | 74.3 |
| | +8.3 | +30.5 | +1.7 | +29.1 | +25.4 | +12.6 | +23.9 |
| GPT-4o | 62.5 | 25.0 | 50.8 | 96.1 | 87.2 | 84.4 | 61.1 |
| + Sketchpad | **66.7** | **66.3** | **65.3** | **98.1** | 90.1 | 88.1 | **81.2** |
| | +4.2 | +41.3 | +14.5 | +2.0 | +2.9 | +3.7 | +20.1 |

Table 1: Accuracy scores on geometry problems, graph algorithms, mathematical functions, and game. **SKETCHPAD yields large performance gains across all tasks and outperform all baselines.**

| Model | $V^*$Bench | MMVP | Depth | Spatial | Jigsaw | Vis. Corr. | Sem. Corr. |
|---|---|---|---|---|---|---|---|
| *Prior multimodal LLMs* | | | | | | | |
| LLaVA-1.5-7B [25] | 48.7 | - | 52.4 | 61.5 | 11.3 | 25.6 | 23.0 |
| LLaVA-1.5-13B [25] | - | 24.7 | 53.2 | 67.8 | 58.0 | 29.1 | 32.4 |
| LLaVA-NeXT-34B [26] | - | - | 67.7 | 74.8 | 54.7 | 30.8 | 23.7 |
| Claude 3 OPUS [1] | - | - | 47.6 | 58.0 | 32.7 | 36.6 | 25.2 |
| Gemini-Pro [38] | 48.2 | 40.7 | 40.3 | 74.8 | 57.3 | 42.4 | 26.6 |
| GPT-4V-preview [34] | 55.0 | 38.7 | 59.7 | 72.7 | 70.0 | 33.7 | 28.8 |
| Previous state of the art | 75.4 [47] | 49.3 [10] | 67.7 [26] | 76.2 [39] | 70.0 [34] | 42.4 [38] | 33.1 [45] |
| *Latest multimodal LLMs + Visual Sketchpad* | | | | | | | |
| GPT-4 Turbo | 52.5 | 71.0 | 66.1 | 68.5 | 64.7 | 48.8 | 30.9 |
| + Sketchpad | 71.0 | 73.3 | 68.5 | 80.4 | 68.5 | 52.3 | 42.4 |
| | +18.5 | +2.3 | +2.4 | +11.9 | +3.8 | +3.5 | +11.5 |
| GPT-4o | 66.0 | 85.3 | 71.8 | 72.0 | 64.0 | 73.3 | 48.6 |
| + Sketchpad | **80.3** | **86.3** | **83.9** | **81.1** | **70.7** | **80.8** | **58.3** |
| | +14.3 | +1.0 | +12.1 | +9.1 | +6.7 | +7.5 | +9.7 |

Table 2: Accuracy on complex visual reasoning tasks. **SKETCHPAD enhances both GPT-4 Turbo and GPT-4o performance, establishing new SOTA performance levels on all the tasks.**

specifically trained for this task. On BLINK tasks, SKETCHPAD on average yields 6.6% absolute accuracy gain for `GPT-4 Turbo` and 9.0% gain for `GPT-4o`. Interestingly, despite the fact that all modules in SKETCHPAD work on a single image, the LMs also get substantial improvement on multi-image tasks, including jigsaw puzzles, visual correspondence, and semantic correspondence. Finally, `GPT-4o`, the LM with stronger multimodal ability than `GPT-4 Turbo`, benefits more from SKETCHPAD. For example, on the relative depth task, `GPT-4o` gets 12.1% accuracy improvement, while `GPT-4 Turbo` only gets 2.4%, showing that `GPT-4o` is better at understanding the depth map SKETCHPAD generated. Overall, our experiments show that SKETCHPAD is an effective way to improve multimodal LMs' performance on visual reasoning tasks. More analysis are in §E.

## 4 Conclusion

We present Visual SKETCHPAD, a framework that provides multimodal LMs with the tools necessary to generate intermediate sketches to reason over tasks. For complex mathematical reasoning tasks, SKETCHPAD yields large performance gains, by visualizing auxiliary lines, math functions, graphs, and games during reasoning. For visual reasoning tasks, we add vision specialists to SKETCHPAD. The LM can call these specialists during reasoning, observing the visualization of these specialists' predictions and then conduct further planning and reasoning. Experiments show that SKETCHPAD enhances the LMs' performance across all tasks, and sets new state-of-the-art results. Ultimately, SKETCHPAD represents a step toward endowing LMs with complementary strengths of language and vision to tackle increasingly complex reasoning challenges.

# References


[1] Introducing the next generation of claude. https://www.anthropic.com/news/claude-3-family, March 2024.

[2] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in Neural Information Processing Systems*, 35:23716–23736, 2022.

[3] Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond, 2023.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[5] Xi Chen, Josip Djolonga, Piotr Padlewski, Basil Mustafa, Soravit Changpinyo, Jialin Wu, Carlos Riquelme Ruiz, Sebastian Goodman, Xiao Wang, Yi Tay, Siamak Shakeri, Mostafa Dehghani, Daniel Salz, Mario Lucic, Michael Tschannen, Arsha Nagrani, Hexiang Hu, Mandar Joshi, Bo Pang, Ceslee Montgomery, Paulina Pietrzyk, Marvin Ritter, AJ Piergiovanni, Matthias Minderer, Filip Pavetic, Austin Waters, Gang Li, Ibrahim Alabdulmohsin, Lucas Beyer, Julien Amelot, Kenton Lee, Andreas Peter Steiner, Yang Li, Daniel Keysers, Anurag Arnab, Yuanzhong Xu, Keran Rong, Alexander Kolesnikov, Mojtaba Seyedhosseini, Anelia Angelova, Xiaohua Zhai, Neil Houlsby, and Radu Soricut. Pali-x: On scaling up a multilingual vision and language model, 2023.

[6] Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Zhong Muyan, Qinglong Zhang, Xizhou Zhu, Lewei Lu, et al. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. *arXiv preprint arXiv:2312.14238*, 2023.

[7] Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale Fung, and Steven Hoi. Instructblip: Towards general-purpose vision-language models with instruction tuning, 2023.

[8] Deqing Fu*, Ghazal Khalighinejad*, Ollie Liu*, Bhuwan Dhingra, Dani Yogatama, Robin Jia, and Willie Neiswanger. IsoBench: Benchmarking multimodal foundation models on isomorphic representations, 2024.

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

[10] Peng Gao, Renrui Zhang, Chris Liu, Longtian Qiu, Siyuan Huang, Weifeng Lin, Shitian Zhao, Shijie Geng, Ziyi Lin, Peng Jin, et al. Sphinx-x: Scaling data and parameters for a family of multi-modal large language models. *arXiv preprint arXiv:2402.05935*, 2024.

[11] Vinod Goel. *Sketches of thought*. MIT press, 1995.

[12] Dirk Groeneveld, Iz Beltagy, Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, et al. Olmo: Accelerating the science of language models. *arXiv preprint arXiv:2402.00838*, 2024.

[13] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962, 2023.

[14] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework, 2023.

[15] Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. Tool documentation enables zero-shot tool-usage with large language models. *arXiv preprint arXiv:2308.00675*, 2023.

[16] Yushi Hu, Hang Hua, Zhengyuan Yang, Weijia Shi, Noah A Smith, and Jiebo Luo. Promptcap: Prompt-guided task-aware image captioning. *arXiv preprint arXiv:2211.09699*, 2022.

[17] Yushi Hu, Otilia Stretcu, Chun-Ta Lu, Krishnamurthy Viswanathan, Kenji Hata, Enming Luo, Ranjay Krishna, and Ariel Fuxman. Visual program distillation: Distilling tools and programmatic reasoning into vision-language models. *arXiv preprint arXiv:2312.03052*, 2023.

[18] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023.

[19] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

[20] Apoorv Khandelwal, Ellie Pavlick, and Chen Sun. Analyzing modular approaches for visual question decomposition. *arXiv preprint arXiv:2311.06411*, 2023.

[21] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4015–4026, 2023.

[22] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024.

[23] Feng Li, Hao Zhang, Peize Sun, Xueyan Zou, Shilong Liu, Jianwei Yang, Chunyuan Li, Lei Zhang, and Jianfeng Gao. Semantic-sam: Segment and recognize anything at any granularity. *arXiv preprint arXiv:2307.04767*, 2023.

[24] Dingning Liu, Xiaomeng Dong, Renrui Zhang, Xu Luo, Peng Gao, Xiaoshui Huang, Yongshun Gong, and Zhihui Wang. 3daxiesprompts: Unleashing the 3d spatial task capabilities of gpt-4v. *arXiv preprint arXiv:2312.09738*, 2023.

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

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

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

[28] Shilong Liu, Hao Cheng, Haotian Liu, Hao Zhang, Feng Li, Tianhe Ren, Xueyan Zou, Jianwei Yang, Hang Su, Jun Zhu, et al. Llava-plus: Learning to use tools for creating multimodal agents. *arXiv preprint arXiv:2311.05437*, 2023.

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

[30] Jiasen Lu, Christopher Clark, Sangho Lee, Zichen Zhang, Savya Khosla, Ryan Marten, Derek Hoiem, and Aniruddha Kembhavi. Unified-io 2: Scaling autoregressive multimodal models with vision, language, audio, and action. *arXiv preprint arXiv:2312.17172*, 2023.

[31] Pan Lu, Ran Gong, Shibiao Jiang, Liang Qiu, Siyuan Huang, Xiaodan Liang, and Song-Chun Zhu. Inter-gps: Interpretable geometry problem solving with formal language and symbolic reasoning. In *The 59th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2021.

[32] Zixian Ma, Weikai Huang, Jieyu Zhang, Tanmay Gupta, and Ranjay Krishna. m&m's: A benchmark to evaluate tool-use for multi-step multi-modal tasks. In *Synthetic Data for Computer Vision Workshop@ CVPR 2024*, 2024.

[33] Soroush Nasiriany, Fei Xia, Wenhao Yu, Ted Xiao, Jacky Liang, Ishita Dasgupta, Annie Xie, Danny Driess, Ayzaan Wahid, Zhuo Xu, Quan Vuong, Tingnan Zhang, Tsang-Wei Edward Lee, Kuang-Huei Lee, Peng Xu, Sean Kirmani, Yuke Zhu, Andy Zeng, Karol Hausman, Nicolas Heess, Chelsea Finn, Sergey Levine, and Brian Ichter. Pivot: Iterative visual prompting elicits actionable knowledge for vlms. 2024.

[34] OpenAI. Gpt-4 technical report, 2023.

[35] Aleksandar Shtedritski, Christian Rupprecht, and Andrea Vedaldi. What does clip know about a red circle? visual prompt engineering for vlms. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11987–11997, 2023.

[36] Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2023.

[37] Chameleon Team. Chameleon: Mixed-modal early-fusion foundation models. 2024.

[38] Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[39] InternLM Team. Internlm: A multilingual language model with progressively enhanced capabilities. `https://github.com/InternLM/InternLM`, 2023.

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

[41] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

[42] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

[43] Barbara Tversky and Masaki Suwa. Thinking with sketches. 2009.

[44] Barbara Tversky, Masaki Suwa, Maneesh Agrawala, Julie Heiser, Chris Stolte, Pat Hanrahan, Doantam Phan, Jeff Klingner, Marie-Paule Daniel, Paul Lee, et al. Sketches for design and design of sketches. *Human Behaviour in Design: Individuals, Teams, Tools*, pages 79–86, 2003.

[45] Weihan Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, Yan Wang, Junhui Ji, Zhuoyi Yang, Lei Zhao, Xixuan Song, et al. Cogvlm: Visual expert for pretrained language models. *arXiv preprint arXiv:2311.03079*, 2023.

[46] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[47] Penghao Wu and Saining Xie. V*: Guided visual search as a core mechanism in multimodal llms. *ArXiv*, abs/2312.14135, 2023.

[48] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023.

[49] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024.

[50] An Yan, Zhengyuan Yang, Wanrong Zhu, Kevin Lin, Linjie Li, Jianfeng Wang, Jianwei Yang, Yiwu Zhong, Julian McAuley, Jianfeng Gao, et al. Gpt-4v in wonderland: Large multimodal models for zero-shot smartphone gui navigation. *arXiv preprint arXiv:2311.07562*, 2023.

[51] Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. *arXiv preprint arXiv:2310.11441*, 2023.

[52] Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything: Unleashing the power of large-scale unlabeled data. In *CVPR*, 2024.

[53] Zhengyuan Yang*, Linjie Li*, Jianfeng Wang*, Kevin Lin*, Ehsan Azarnasab*, Faisal Ahmed*, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. Mm-react: Prompting chatgpt for multimodal reasoning and action. 2023.

[54] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

[55] Andy Zeng, Maria Attarian, Brian Ichter, Krzysztof Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, et al. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*, 2022.

[56] Jieyu Zhang, Ranjay Krishna, Ahmed H Awadallah, and Chi Wang. Ecoassistant: Using llm assistant more affordably and accurately. *arXiv preprint arXiv:2310.03046*, 2023.

[57] Zhuosheng Zhang, Aston Zhang, Mu Li, Hai Zhao, George Karypis, and Alex Smola. Multimodal chain-of-thought reasoning in language models. *arXiv preprint arXiv:2302.00923*, 2023.

# Appendices

# A Related Work

SKETCHPAD generalizes recent work on multimodal tool-use and visual prompting. We also place our work within the larger sphere exploring LMs as agents.

**Visual programming and tool-use.** With the advancement of LMs [4, 34, 38, 42, 12], researchers have demonstrated the possiblity of decomposing complex vision tasks into simpler substeps that can each be solved using vision tools [53, 55, 17, 16]. Among them, the most relevant to us are Visprog [13] and ViperGPT [36]. They use LMs to generate Python code, which sequencially invokes specialized vision tools. These methods share a common problem that the multimodal modules follow a pre-defined plan outlined by the LM. By contrast, SKETCHPAD allows LMs to change their plan according to the intermediate visual artifacts, yielding better performance and robustness when solving complex multimodal tasks.

**Visual prompting.** Recent work shows that multimodal models can be augmented by visual prompts added to natural images [35]. For example, SoM [51] shows that adding labeled segmentation masks on images unleashes GPT-4V's visual grounding ability. Prior work also reports similar findings in 3D [24] and Robotics [33]. SKETCHPAD is a generalized framework for all these methods, allowing LMs to decide what visual prompting to use as part of the multimodal reasoning process.

**LMs as agents.** Recent work has started to treat LMs as agents that can both reason and act [54, 32, 48]. Researchers have applied this idea to software engineering [19, 56, 14], robotics [33], vision [28, 53], and GUI navigation [50, 22, 49]. SKETCHPAD can also be viewed as an agent that accepts multimodal inputs and outputs. One big difference is that SKETCHPAD can create visual artifacts to facilitate reasoning, while prior LM agents only generate texts during reasoning.

# B Sketching via Code Generation

The core component of SKETCHPAD is sketching, which enables the LM to generate visual sketches by synthesizing programs that call different specialist vision models or Python plotting packages.

**Program Generation.** Similar to recent works like ViperGPT and VPD [13, 36, 17], SKETCHPAD enables LMs to sketch through code generation. The LM is provided, through a prompt, with a detailed description of the available tools that can generate multimodal content (an example prompt and description can be found in §F). The prompt includes Python function signatures and docstrings [15] for these modules, but does not contain their full implementation. The LM generates Python code in a code block, using the provided tools, which, when executed, generates new image and text outputs. A special *display* function allows the LM to **visualize** the sketch image in the next observation $o_{t+1}$.

**Modules for sketching.** SKETCHPAD uses various tools to facilitate the sketching process, depending on the task at hand. For mathematical tasks, SKETCHPAD uses common Python packages like `matplotlib` and `networkx` for plotting (see §3). For vision tasks, the LM leverages **specialist vision models** during the sketching process. These models include detection tools that draw bounding boxes on the image, as well as segmentation and marking tools (inspired by SoM [51]) that draw colorful masks on the image and use numbers to label segments. We find these specialists possess essential perception skills for visual reasoning tasks, and SKETCHPAD is an effective way to combine them into a multimodal LM (see §D.1).

# C Math tasks details

**Geometry Problems.** Drawing auxiliary lines in geometry diagrams is often helpful for problem-solving. For example, in Figure 2 (a), when asked to find $\angle EIC$, the LM plans to draw an auxiliary line $IX$ parallel to $BD$, allowing it to use the properties of parallel lines to determine $\angle EIC$. To evaluate the effectiveness of SKETCHPAD, we use the problems from the Geometry3K dataset [31].

To realize the line drawing process, SKETCHPAD takes a geometry diagram and its corresponding `matplotlib` code as input. The model then proposes and modifies the code to generate auxiliary lines, and executes the modified code to visualize the updated diagram with the added lines.

**Mathematical functions.** Understanding mathematical functions is crucial for various applications in science, engineering, and economics. We focus on two tasks related to mathematical functions from the IsoBench datasets [8]:

- ***Classifying parity*** aims to determine whether a function is even, odd, or neither. Even functions satisfy $f(-x) = f(x)$ for all $x$, while odd functions satisfy $f(-x) = -f(x)$.
- ***Identifying convexity/concavity*** aims to determine whether a function is convex or concave.

Existing LMs can only analyze functions and attempt to prove their properties analytically. [1] However, SKETCHPAD enables them to visually sketch functions to solve problems more efficiently. For instance, to determine the convexity of the function in Figure 1b, SKETCHPAD allows the model to plot the function using `matplotlib`, and visually inspect its overall shape.

**Graph algorithms.** Many real-world problems, such as those related to computer networks and transportation systems, can be formulated as graph problems. We evaluate SKETCHPAD on three graph problems from IsoBench [8]:

- ***Graph connectivity*** determines whether there exists a path between two vertices in a graph.
- ***Maximum flow*** aims to find the maximum amount of flow that can be sent through a network from a source vertex to a sink vertex, subject to capacity constraints on the edges.
- ***Graph isomorphism*** tests whether two graphs are structurally equivalent.

Given an adjacency matrix of a graph like in Figure 2(b), SKETCHPAD can draw the actual graph structure, using using Python's `networkx` library, enabling direct visual reasoning about graph properties and relationships.

**Game strategies.** Chess games can be represented in various formats, including visual board states and textual move notations. Given only the textual move notations, SKETCHPAD can draw the visual representations of the chess board to analyze positions and formulate strategies. We evaluate the performance of SKETCHPAD on the winnder identification task from the IsoBench datasets [8] that aims to find the outcome of a chess game (win for White, win for Black, or draw) based on the final board state. To create the graphical board, SKETCHPAD uses Python's `chess` library to draw the board from the Forsyth-Edwards Notation (FEN) of chess.

# D Computer Vision tasks details

**Tasks.** We experiment with a wide range of complex visual reasoning tasks: (1) $V^*$**Bench** [47]. This benchmark contains questions about small items in an image. (2) **MMVP** benchmark from *Eyes Wide Shut* [40]. This benchmark contains visual questions specially designed to reveal the visual shortcomings of CLIP-based multimodal LMs. (3) **BLINK** [9]. This benchmark contains visual perception tasks that are easy for humans, but post significant challenge for multimodal LMs. Specifically, we experiment with relative depth, spatial reasoning, jigsaw puzzle, visual correspondence, and semantic correspondence tasks. More details of each task are in §G.

## D.1 Vision Specialists as Sketching Tools in SKETCHPAD

LMs can use the following modules to sketch and manipulate images. We wrap these modules into Python functions that the LMs can call. Refer to §F for the function definitions.

**Detection.** This module takes an image and a simple text query (e.g., "cat") as input. We run the Grounding-DINO [29] open-vocabulary objection detection model and plot the detected bounding boxes (together with a number label) on the image. It also returns the bounding box coordinates.

**Segmentation.** This module takes an image as input and returns an image with colorful segmentation masks on it. Each mask also has a number label. We follow the implementation of SoM [51]. The underlying segmentation models are SegmentAnything [21] and Semantic-SAM [23].

**Depth estimation.** This module takes an image as input and returns a depth map. The underlying model is DepthAnything [52].

[1] For humans, the analytical approach is the correct way to tackle these tasks. However, we observe that LMs are not good at analytical reasoning in math. They make errors when deducing $f(-x)$ and derivatives.

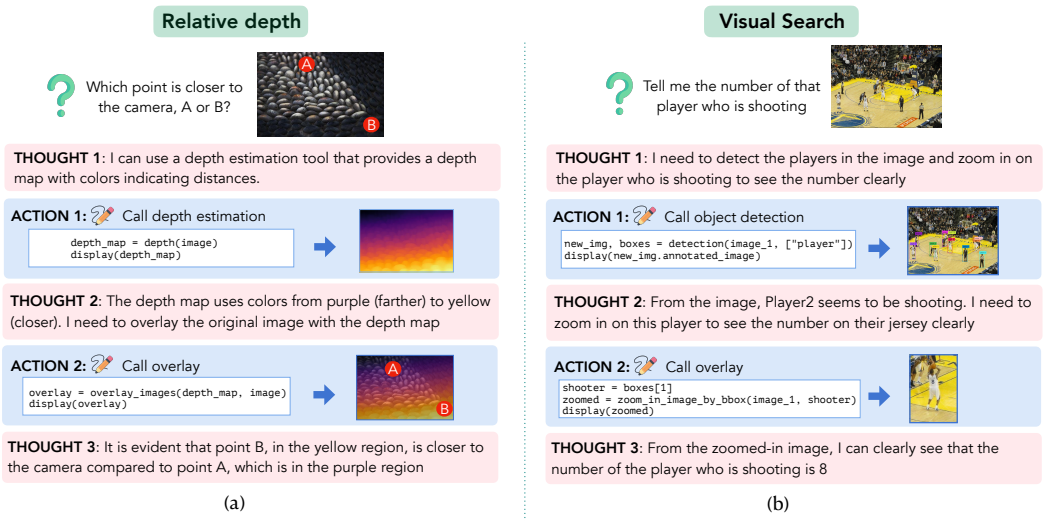


Figure 3: Examples of SKETCHPAD applied to vision tasks. The figure shows actual outputs generated by SKETCHPAD. By contrast, the baseline GPT-4o model cannot answer these questions correctly.

**Visual search via sliding window.** This module mimics how humans search for small items on an image. It takes a text query as input and runs a sliding window over the image. The window size is 1/3 of the image size, and the step size is 2/9 of the image size (so an image will have $4 \times 4 = 16$ windows). It returns a sequence of image patches in which the query is detected.

**Other image manipulation modules.** Other modules include (1) **zoom-in and crop**, which takes an image and a bounding box as input and returns the image patch inside the box; (2) **Overlay images**, which takes two images and alpha values as input, and returns the overlayed image.

# E More Analysis

**How many times is each vision specialist used?** We count the number of times each vision specialist is used in each task, as shown in Figure 4. Here we choose the four tasks that achieve the largest improvement: $V^*$Bench, relative depth, spatial reasoning, and semantic correspondence. We observe that (1) **the use of vision specialist is task-dependent, and the two LMs analyzed utilize similar tools.** For example, for $V^*$, which needs to locate small objects, the LMs mainly use detection, sliding window search, and zoom-in, similar to how people would search. For the relative depth task, both models rely on depth estimation. For spatial reasoning, the LMs use detection and segmentation to facilitate visual reasoning. (2) **GPT-4o likes to use more tools.** `GPT-4o` uses the vision specialists more often than `GPT-4 Turbo`. Also, the two LMs behave differently for the semantic correspondence tasks. `GPT-4o` uses the segmentation module for 40% of the task instances, while `GPT-4 Turbo` uses the detection module for less than 20% of times, and rarely use the segmentation module. This difference may explain the performance gap between the two LMs (58.3% v.s. 42.4%) on this task.

**Comparison with visual prompting and tool-use frameworks.** In Table 3, we compare SKETCHPAD with the visual prompting framework **SoM** [51] and the LLM tool-use framework **Visprog** [13]. Details of these methods can be found in §A. For a fair comparison, we make the following adaptations: (1) we find that prompting LMs with SoM images can hurt performance, likely because the visual prompts confuse the model. To make a stronger baseline, we prompt the LM with both the original image and the SoM image (full prompt in §F), which we refer as "SoM + orig." (2) We replace the LM and VQA modules in Visprog with the corresponding GPT-4 model. (3) Since baseline methods are developed on single-image tasks, we compare SKETCHPAD on such tasks. From Table 3, we can see that **SKETCHPAD is the only framework that yields consistent improvement on all tasks.** SoM can boost spatial reasoning ability, as the authors reported. However, it can hurt the performance on other tasks, even in the "SoM + orig." setting. Visprog performs worse than the base LM on all the tasks. As prior work [20, 17] suggests, one possible reason is that the vision modules themselves have errors, and the error propagates when the modules are composed by a program.

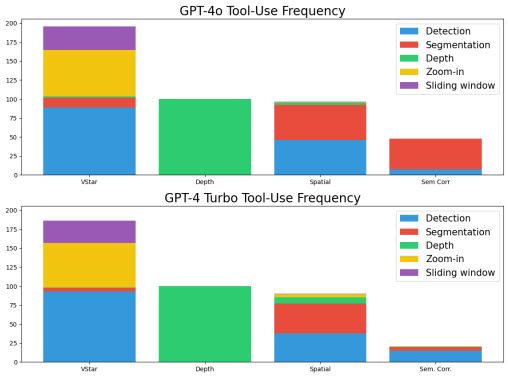


Figure 4: Percentage of times GPT-4o and GPT-4 Turbo use a visual module in SKETCHPAD when solving $V^*$Bench, relative depth, spatial reasoning, and semantic correspondence tasks.

| Model | $V^*$ | MMVP | Depth | Spatial |
|---|---|---|---|---|
| GPT-4 Turbo | 52.5 | 71.0 | 66.1 | 68.5 |
| SoM | 42.0 | 60.7 | 58.9 | 78.3 |
| SoM + orig. | 51.3 | **74.3** | 66.9 | 79.7 |
| Visprog | 33.2 | 16.3 | 67.8 | 53.8 |
| Sketchpad | **71.0** | 73.3 | **68.5** | **80.4** |
| GPT-4o | 66.0 | 85.3 | 71.8 | 72.0 |
| SoM | 49.0 | 70.7 | 62.9 | **83.2** |
| SoM + orig. | 68.1 | 84.0 | 75.0 | 82.5 |
| Visprog | 32.4 | 17.3 | 46.8 | 37.8 |
| Sketchpad | **80.3** | **86.3** | **83.9** | 81.1 |

Table 3: Comparison with other augmentation frameworks for multimodal LMs on single-image tasks. For fair comparison, we modify the original Visprog [13] framework by replacing the LM and VQA components with the corresponding GPT-4 model.

**Why does SKETCHPAD work?** First, **vision is a versatile and informational interface that complements language**. Dense information like depth and segmentation cannot be described easily through language [9]. In a broader perspective, humans have developed many visualization techniques that are direct, efficient, and informational. SKETCHPAD provides LMs the opportunity to use them. Second, in SKETCHPAD, multimodal LMs can **plan and reason based on the intermediate visual artifacts** they created. In contrast, in prior modular vision work [13, 36, 51], multimodal modules follow a predefined plan by either humans or code. SKETCHPAD is much more flexible and robust to errors. For example, suppose object detection makes an error. The LM can (in principle) find the error by viewing the bounding boxes, and change its following plans, but prior methods cannot. Third, as discussed next, **the plans of multimodal LMs are similar to human plans**, and therefor likely benefit from the fact that the underlying LMs have seen data with similar reasoning patterns during pretraining.

**Do LMs have the same plans as humans?** We conduct a human study on all geometry problems and 10 problems on each vision task. On geometry, humans draw the same auxiliary line as GPT-4o 80% of the time. On vision, we show 2 human subjects the full plan of GPT-4o, which they rate is valid in 92.8% of instances. Most errors are caused by failures in the vision specialists (e.g., fail to detect an object) and mistakes in simple visual questions answering, rather than planning.

| Model | Geometry | Maxflow | Convexity | Winner ID |
|---|---|---|---|---|
| LLaVA-NeXT-13B | 11.1 | 7.8 | 50.39 | 5.8 |
| + oracle Sketchpad | 22.2 | 10.2 | 50.0 | 36.7 |
| LLaVA-NeXT-34B | 26.1 | 0.8 | 81.6 | 49.0 |
| + oracle Sketchpad | 28.3 | 14.1 | 87.1 | 49.4 |

Table 4: Open-source LLaVA models' performance on math tasks. The oracle Sketchpad uses the visual artifact generated in the last action of GPT-4o + SKETCHPAD as inputs.

**Experiments on open-source models.** Can sketches like diagrams, plots, and auxiliary lines facilitate existing open-source multimodal LMs? To answer this question, we conduct the experiments in Table 4. We use the visual artifacts generated in the last action of GPT-4o + SKETCHPAD experiment as the image input for open-source LLaVA-NEXT models [26]. We can see that this oracle SKETCHPAD brings consistent improvement to math tasks and boosts mathematical reasoning.

# F Prompts

- **Code**: 1. merge math into Yushi' code ]$_{-\text{WS}}$

Here we provide our prompts for math tasks as in Figures 5 and 6.

PROMPT
You are given a real-valued, scalar function $f(x)$.
YOUR TASK is to determine whether $f(x)$ is an convex function or concave function.
Definition of a convex function: A function such that

$$\forall x, y, 0 \leq t \leq 1, f(tx + (1-t)y) \leq tf(x) + (1-t)f(y)$$

Definition of a concave function: A function such that

$$\forall x, y, 0 \leq t \leq 1, f(tx + (1-t)y) \geq tf(x) + (1-t)f(y)$$

Here is the expression of $f(x)$, defined for all x>0. Here is the expression of $f(x)$:

$$f(x) = 7.57 - 0.08 * Abs(x)$$

Respond with 'convex' or 'concave' first on whether the function f (x) is convex or concave, based on the definitions and your observation of the function. You can generate matplotlib code to visualize the function.
If you can get the answer, please reply with ANSWER: <your answer>, extract the final answer in FINAL ANSWER: <final answer> and ends with TERMINATE in the RESULT.
Answer:

Figure 5: Prompt for the Math Convexity task. We follow the similar prompt format to [8], except prompting the models to write the code to generate images.

PROMPT
You are given an adjacency matrix of a graph and two query nodes.
YOUR TASK is to find if there is a path between the two nodes.
Definition of connectivity: In an undirected graph G, two vertices u and v are called connected if G contains a path from u to v. A path in a graph is a finite sequence of edges which joins a sequence of vertices.
In the query example, the nodes and the adjacency matrix are zero-indexed.
Query Example:
Adjacency Matrix:
[[0, 0, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 1, 0, 0],
[0, 1, 0, 0, 1, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 1, 0, 0, 0, 0, 0, 0],
[1, 0, 0, 0, 0, 0, 1, 0, 1],
[0, 1, 0, 0, 0, 1, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 1, 0, 0, 0]]
Query nodes indices (zero-indexed): 4 and 0
Respond with 'yes' or 'no' first on whether the query nodes are connected or not in the graph.
If there is a path, first provide the path as a sequence of vertices (nodes), and then explain your reasoning. You can use networkx to draw the graph. If there is no path, explain why in details.
Answer (start with 'yes' or 'no'): If you can get the answer, please reply with ANSWER: <your answer>, extract the final answer in FINAL ANSWER: <final answer> and ends with TERMINATE in the RESULT.
Answer:

Figure 6: Prompt for the Graph Connectivity task. We follow the similar prompt format to [8], except prompting the models to write the code to generate images.

Here we provide the prompts for GPT-4o + SKETCHPAD tackles a visual search problem. For simplicity, we omitted the coding parts for tools besides detection and sliding window.

```
SYSTEM PROMPT: You are a helpful multimodal AI assistant.
Solve tasks using your vision, coding, and language skills.
The task can be free-form or multiple-choice questions.
You can answer the user's question about images.  If you are not sure, you can
coding
You are coding in a Python jupyter notebook environment.
You can suggest python code (in a python coding block) for the user to execute.  In
a dialogue, all your codes are executed with the same jupyter kernel, so you can use
the variables, working states..  in your earlier code blocks.  Solve the task step
by step if you need to.
The task may be a vision-language task and require several steps.  You can write
code to process images, text, or other data in a step.  Give your code to the
user to execute.  The user may reply with the text and image outputs of the code
execution.  You can use the outputs to proceed to the next step, with reasoning,
planning, or further coding.
If a plan is not provided, explain your plan first.  Be clear which step uses code,
and which step uses your language skill.
When using code, you must indicate the script type in the code block.  The user
cannot provide any other feedback or perform any other action beyond executing the
code you suggest.  The user can't modify your code.  So do not suggest incomplete
code which requires users to modify.  Don't use a code block if it's not intended to
be executed by the user.
Don't include multiple code blocks in one response.  Do not ask users to copy and
paste the result.  Instead, use 'print' function for the output when relevant.
Check the execution result returned by the user.
All images should be stored in PIL Image objects.  The notebook has imported 'Image'
from 'PIL' package and 'display' from 'IPython.display' package.  If you want to
read the image outputs of your code, use 'display' function to show the image in the
notebook.  The user will send the image outputs to you.
If the result indicates there is an error, fix the error and output the code again.
Suggest the full code instead of partial code or code changes.  If the error can't
be fixed or if the task is not solved even after the code is executed successfully,
analyze the problem, revisit your assumption, collect additional info you need, and
think of a different approach to try.
For each turn, you should first do a "THOUGHT", based on the images and text you
see.  If you think you get the answer to the intial user request, you can reply with
"ANSWER: <your answer>" and ends with "TERMINATE".



ROLE: User:
Here are some tools that can help you.  All are python codes.  They are in tools.py
and will be imported for you.
The images has their own coordinate system.  The upper left corner of the image is
the origin (0, 0).  All coordinates are normalized, i.e., the range is [0, 1].
All bounding boxes are in the format of [x, y, w, h], which is a python list.  x is
the horizontal coordinate of the upper-left corner of the box, y is the vertical
coordinate of that corner, w is the box width, and h is the box height.
Notice that you, as an AI assistant, is not good at locating things and describe
them with coordinate.  You can use tools to generate bounding boxes.
You are also not good at answering questions about small visual details in the
image.  You can use tools to zoom in on the image to see the details.  Below are the
tools in tools.py:
```

```python
class AnnotatedImage:
# A class to represent an annotated image.  It contains the annotated image and the
original image.
def __init__(self, annotated_image:  Image.Image, original_image:
Image.Image=None):
self.annotated_image = annotated_image
self.original_image = original_image


def detection(image, objects):
"""Object detection using Grounding DINO model.  It returns the annotated image and
the bounding boxes of the detected objects.
The text can be simple noun, or simple phrase (e.g., 'bus', 'red car').  Cannot be
too hard or the model will break.
The detector is not perfect, it may wrongly detect objects or miss some objects.
Also, notice that the bounding box label might be out of the image boundary.
You should use the output as a reference, not as a ground truth.
When answering questions about the image, you should double-check the detected
objects.
Args:
image (PIL.Image.Image):  the input image
objects (List[str]):  a list of objects to detect.  Each object should be a simple
noun or a simple phrase.  Should not be hard or abstract concepts like "text" or
"number".
Returns:
output_image (AnnotatedImage):  the original image, annotated with bounding boxes.
Each box is labeled with the detected object, and an index.
processed boxes (List):  listthe bounding boxes of the detected objects
Example:
image = Image.open("sample_img.jpg")
output_image, boxes = detection(image, ["bus"])
display(output_image.annotated_image)
print(boxes) # [[0.24, 0.21, 0.3, 0.4], [0.6, 0.3, 0.2, 0.3]]
# you need to double-check the detected objects.  Some objects may be missed or
wrongly detected.
"""


def sliding_window_detection(image, objects):
"""Use this when you are searching for objects in the image, but the objects are not
detected by the object detection model.
In that case, the most common reason is that the object is too small such that both
the vision-language model and the object detection model fail to detect it.
This function tries to detect the object by sliding window search.
With the help of the detection model, it tries to detect the object in the zoomed-in
patches.
The function returns a list of annotated images that may contain at leas one of the
objects, annotated with bounding boxes.
It also returns a list of a list of bounding boxes of the detected objects.
Args:
image (PIL.Image.Image):  the input image
objects (List[str]):  a list of objects to detect.  Each object should be a simple
noun or a simple phrase.  Should not be hard or abstract concepts like "text" or
"number".
Returns:
possible_patches (List[AnnotatedImage]):  a list of annotated zoomed-in images that
may contain the object, annotated with bounding boxes.
possible_boxes (List[List[List[Float]]]):  For each image in possible_patches, a
list of bounding boxes of the detected objects.
The coordinates are w.r.t.  each zoomed-in image.  The order of the boxes is the
same as the order of the images in possible_patches.  ```
```

```
# GOAL #:  Based on the above tools, I want you to reason about how to solve the
# USER REQUEST # and generate the actions step by step (each action is a python
jupyter notebook code block) to solve the request.
You may need to use the tools above to process the images and make decisions based
on the visual outputs of the previous code blocks.
Your visual ability is not perfect, so you should use these tools to assist you in
reasoning about the images.
The jupyter notebook has already executed the following code to import the necessary
packages:
‘‘‘python
from PIL import Image
from IPython.display import display
from tools import *
‘‘‘
# REQUIREMENTS #:
1.  The generated actions can resolve the given user request # USER REQUEST #
perfectly.  The user request is reasonable and can be solved.  Try your best to
solve the request.
2.  The arguments of a tool must be the same number, modality, and format specified
in # TOOL LIST #;
3.  If you think you got the answer, use ANSWER: <your answer> to provide the
answer, and ends with TERMINATE.
4.  All images in the initial user request are stored in PIL Image objects named
image_1, image_2, ..., image_n.  You can use these images in your code blocks.  Use
display() function to show the image in the notebook for you too see.
5.  Use as few tools as possible.  Only use the tools for the use cases written in
the tool description.  You can use multiple tools in a single action.
6.  You must return an answer with the choice letter if the user request is a
multiple-choice question.
7.  When detection tool failed to detect an object, you can use the
sliding_window_detection tool to search for the object.
8.  Bounding boxes may be wrong and misled you.  When answering questions about
small objects in bounding boxes, you should zoom in on the object to see the
details.
9.  Segmentation and marking tool can help you better reason about the
relationship between objects.  Example use cases are spatial reasoning (e.g.,
left/right/above/below/on) and counting.


[Examples]

USER REQUEST :
```

Figure 7: The first-turn inputs we give to the LLM for computer vision tasks

## G Dataset statistics

Table 5 and 6 show the statistics of the datasets we used, including IsoBench [8], BLINK [9], MMVP [40], and $V^*$Bench [47].

| Dataset | size | partition | representation |
|---|---|---|---|
| Math Parity | 383 | val | code |
| Math Convexity | 255 | val | code |
| Graph Maxflow | 128 | val | array |
| Graph Connectivity | 128 | val | array |
| Graph Isomorphism | 128 | val | array |
| Winner ID | 257 | val | FEN |

Table 5: IsoBench [8] data statistics.

| Dataset | size | partition | input |
|---|---|---|---|
| $V*$Bench | 257 | - | Single Image |
| MMVP | 300 | - | Single Image |
| BLINK Relative Depth | 124 | val | Single Image |
| BLINK Spatial Relation | 143 | val | Single Image |
| BLINK Jigsaw Puzzle | 150 | val | Multiple Images |
| BLINK Visual Correspondence | 172 | val | Multiple Images |
| BLINK Semantic Correspondence | 139 | val | Multiple Image |

Table 6: Vision tasks data statistics.

## H Costs

The cost of running each task using GPT-4o is in Table 7.

| Dataset | tokens per sample | GPT-4o cost per sample |
|---|---|---|
| Math Parity | 2994 | $0.015 |
| Math Convexity | 2211 | $0.011 |
| Graph Connectivity | 2819 | $0.014 |
| Graph Isomorphism | 3143 | $0.016 |
| $V^*$Bench | 26647 | $0.133 |
| MMVP | 11870 | $0.059 |
| BLINK Relative Detph | 14078 | $0.070 |
| BLINK Spatial Relation | 12848 | $0.064 |
| BLINK Jigsaw Puzzle | 13206 | $0.066 |
| BLINK Visual Correspondence | 16988 | $0.085 |
| BLINK Semantic Correspondence | 11508 | $0.058 |

Table 7: The cost of running SKETCHPAD on each task.