MMO: An Investigation of Multi-modal Multi-agent Organization and Robust Benchmarking

Simon C. Wang*
Department of Computer Science
University of Maryland
College Park, MD
scwang00@umd.edu

Abstract

With the increase of new multimodal large language models (MLLMs) being released with incredible results across various benchmarks, it has become more important than ever before to understand how we actually compare these amazing models. Most standard benchmarks used to demonstrate the capabilities of recent MLLMs have been curated datasets based on high level (college and beyond) questions, expert generated problems, and even screenshots of images from textbooks or webpages. Initially aiming to develop a multi-agent system to leverage the power of small open-source MLLMs, I now present an investigation into the shortcomings of popular benchmark evaluation, as well as a discussion of how it might be improved. In the process of evaluating benchmarks for various models as well as a multi-agent framework developed by myself, I also present a flexible system designed to test one or more MLLMs on common multimodal benchmarks. All code used to generate results can be found at https://github.com/simoncwang/MMO.

1 Introduction

Recent MLLMs such as OpenAI's gpt-4o, and Anthropic's Claude 3, have shown incredible results. They seem to be capable of understanding both images and text to a deep level, answer complex reasoning questions across all of the top benchmarks. However, these models can often struggle with using new information, and maintaining knowledge across multiple modalities over time [9]. In order to improve complex reasoning performance, various strategies are used. First, many prompting strategies aim to encourage models to "think" in various ways similar to how humans process a complex task. For example, [10] found that simply asking the model to think step by step could result in significant reasoning improvements. OpenAI's o1 model [8] take it one step further and uses deep reinforcement learning to force the model to think in this way beyond simply prompting. Another direction is using multi-agent systems, where various models act as agents to solve parts of a complex task in order to hopefully produce a single robust answer [2].

Inspired by the idea of organizing multiple models, I approached the problem of how to orchestrate these agents to best answer complex multimodal problems. Noticing how previous strategies defined rigid roles for each agent, I attempted to create a dynamic subtask-assignment system, using a single powerful model as an orchestrator or "commander" to delegate multiple smaller open-source models.

However, throughout the process of developing this system, I was faced with a more significant challenge than simply implementing a model organization framework. In order to test the effectiveness of my system, I had to first decide what benchmarks were the most suitable for the multimodal

*Final project for CMSC848K - Multimodal Foundation Models course at University of Maryland taught by Professor Jia-Bin Huang

Preprint. Under review.

nature of the models. But, throughout the process of implementing scripts to evaluate my models on these datasets, I became more aware of the inconsistency and complexity of the benchmarking space. Successfully evaluating even a base open-source model available from Huggingface involved extensive manual prompt tuning and often arbitrary output format parsing. So, in addition to developing a relatively simple proof-of-concept for a multi-modal multi-agent system, the majority of my time was spent experimenting with improving benchmark performance and developing a system to test such agentic systems. Alongside some benchmark results on two standard MLLM evaluation datasets, this work will involve an in-depth exploration of current evaluation approaches and a discussion of how benchmarking can be improved overall.

2 Related Works

Various research has also attempted to improve model reasoning ability through multi-agent organization. [4] prompted multiple LLMs to propose and debate individual responses over multiple rounds, showing promising improvements over previous single-agent methods. CoMM [2] explored organizing multiple agents by assigning expert roles and defining different reasoning paths for each role. Similarly, MetaGPT [5] encodes standardized operating procedures into prompt sequences, explicitly assigning software engineering related roles to several agents. Recently, OpenAI's o1 model [8] used reinforcement learning to teach their model to think using a chain of thought approach. This approach modeled after human reasoning yielded significant improvement over previous state of the art on benchmarks like competition math and coding.

There are also many standard benchmarking datasets used for MLLM evaluation and comparison. The datasets considered for this investigation are question-answer (QA) problem sets, mostly consisting of questions with images and multiple choice options across a wide variety of subjects. To evaluate my models, I chose ScienceQA [7], which is collected from elementary and high school level science curricula, as well as MMMU (Massive Multi-discipline Multimodal Understanding) [11] which is gathered from college level questions. MMMU has been one of the most commonly used benchmarks for recent state-of-the-art models, alongside other similar datasets such as ChartQA, OK-VQA, and OpenVQA among others [6].

3 Method

As mentioned prior, the method of this work will have two primary portions. First, I will explain the overall implementation and structure of the MMO (multi-modal multi-agent organization) system. MMO was initially intended to be a complex system with a dynamic subtask reallocation system as well as a graph-based task storage system. However, due limitations of resources such as GPU memory limits and runtime, the system has been converted into an environment primarily geared towards observing the effect of various factors on benchmark performance. The second portion will be a deeper dive into the experiments conducted on the small suite of models originally selected for the multi-agent system, as well as implementation details of improvements made to existing evaluation code.

3.1 MMO: A Multi-modal Multi-agent Organization Framework

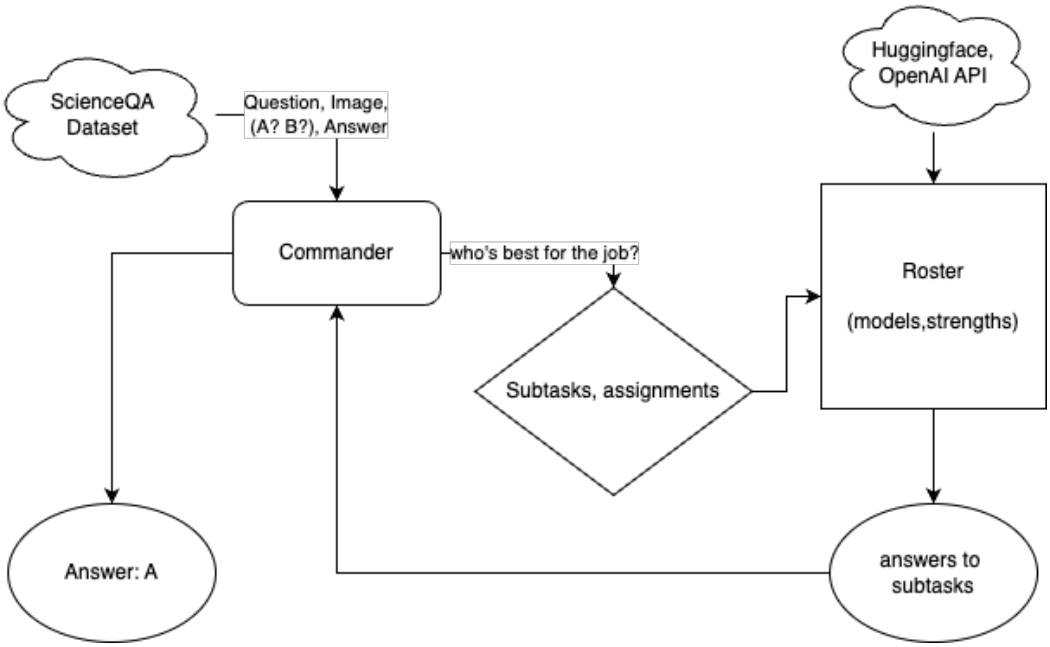
3.1.1 Implementation

MMO consists of two Python classes: Commander, and Roster. The Commander is the overall orchestrator of the system, and is powered by the most capable model. In this case, I use gpt-4o because of OpenAI's new structured outputs feature which allows me to define classes that the model will absolutely adhere to. In this case, the commander model is prompted to break down each question in the dataset and assign it to the agent best suited for the job.

To facilitate this, the Roster class stores a list of all available models in the system, along with a short identifying "codename" that describes its strengths. For example, InternVL is tagged "vision-highquality-slower". The Roster also handles intializing all models in its list, including loading models from Huggingface or setting up OpenAI clients.

As shown in Figure 1, the models in the roster then return their answers to the subtasks, which is used as context by the commander model to give a single final answer.

Figure 1: Structure of the MMO system



3.1.2 Prompting

As mentioned earlier, the overall MMO system is based on the following classes which are used as structured outputs for gpt-4o.

```
# a single subtask
class Subtask(BaseModel):
    subtask: str
    assigned_model: str

# list of all subtasks
class Subtasks(BaseModel):
    original_question: str
    subtasks: list[Subtask]

class ScienceQAAnswer(BaseModel):
    answer: int

class MMMUAnswer(BaseModel):
    answer: str
```

To prompt gpt-4o to act as a coordinator, the following prompts are used:

```
# generating subtasks
system_prompt = f"You are an expert manager, based on the following list of
avialable model names and their strengths: {model_strengths}, assign subtasks
to each model that will help solve the overall question and can be solved
INDEPENDENTLY. Do NOT answer the question yourself, and return the model name
EXACTLY as provided. You can create more subtasks than the number of models,
and/or assign the same type of model to multiple subtasks. If there is no
image provided, do NOT assign any tasks involving analyzing an image."

# providing final answer based on subtask context
system_prompt = "You are an expert at answering multiple choice questions
```

```
given answers to subtasks as context"
prompt = f"Answer the following question: {question_prompt} by providing
the {answer_format} of the correct choice. Use the image if provided as well
as the following answers to subtasks to inform your final answer:
{subtask_context}"
```

3.1.3 Majority Voting

Additionally, within the MMO framework I have implemented a majority voting option as opposed to the automatic subtask assignment system mentioned above. Currently, this option makes multiple calls (5 used here) using a single model, then picks the response with the highest number of votes. This effectively forces the model to slow down and have less opportunity to make a hallucinated answer choice.

3.2 Model Selection

I selected models based on the following criteria; the model has to be open-source and lightweight enough so that I can experiment with it locally, the model has to have proven vision capabilities, and finally the model has to have publicly available code for inference and evaluation. Looking into the sizes of common models as well as the latest most downloaded MLLMs on Huggingface, I narrowed down my selection to two families of models, Qwen and Intern. Both QwenVL [1] and InternVL [3] utilize vision transformers combined with a pre-trained LLM through either an adapter or contrastive training respectively. Both of these models have been shown to be highly competitive with both closed and open-source state-of-the-art across many benchmarks such as MMMU and ScienceQA, despite having fewer parameters and higher efficiency claims.

Looking into typical model weight memory requirements, I chose Qwen2-VL-2B-Instruct and InternVL-2.5-2B as my primary vision language MLLMs (with B indicating billions of parameters). I also use Qwen2.5-1.5B, which is a text-only model, as comparison alongside the other MLLMs.

3.3 Benchmarks

3.3.1 Existing Methods

As mentioned previously, I chose ScienceQA and MMMU as the benchmarks for evaluating my models. In order to establish a baseline, I first attempted to look online for any publicly available evaluation code. Unfortunately, reading more into the Qwen and Intern papers as well as the official GitHub repositories, there was minimal code available to reproduce their results. However, on the ScienceQA GitHub I discovered code for evaluating GPT3 on the dataset, and decided to use this script as the base for my own code. Quickly, I discovered many challenges in implementing these evaluations. Under the hood, typical evaluation results are achieved through very extensive prompt engineering and brute force regular expression pattern matching. I also investigate other popular benchmarks such as GPQA (graduate level QA) and found very similar results.

For example, in the original ScienceQA evaluation code, the standard prompt actually includes an in-context example to help the model follow an expected format, such as: *"output = f"Answer: The answer is {answer}.""*. Similarly, in the GPQA evaluation code, the model is simply asked nicely to return a format as follows: *"After your reasoning, provide your final, single letter choice, formatted as "The answer is (X)""*.

After some trial and error to successfully run the original scripts, I achieved very low accuracy scores, especially for the open source models with Qwen2-VL-2B getting a score of 18.70%. Meanwhile, gpt-4o and gpt-4o-mini both achieved socres of over 85%. Upon closer inspection, the smaller MLLMs were producing many "invalid" results, with Qwen2-VL-2B producing over 700+. However, in many instance the results being produced are actually correct, but just not following the exact format expected by the evaluation code (e.g. *"The answer is A"*), instead returning a single letter or even a different format like *"Choice A*.

Looking deeper into repositories such as GPQA, rather than any deep approaches to improving the model, these high error rates are simply mitigated by providing a wide net to capture potential answer formats in the form of regular expression patterns. For example, here is the pattern list used in GPQA:

```
def parse_sampled_answer(answer, answer_choice_tokens):
    """Copy pasted from GPQA repo + more patterns"""
    patterns = [
        r"answer is \((.)\)",
        r"Answer: \((.)\)",
        r"answer: \((.)\)",
        r"answer \((.)\)",
        r"answer is (\w)\.",
        r"Answer: (\w)\.",
        r"answer: (\w)\.",
        r"answer (\w)\.",
        r"answer is option \((.)\)",
        r"answer is Option \((.)\)",
        r"answer is option (\w)",
        r"answer is Option (\w)",
    ]
    for pattern in patterns:
        match = re.search(pattern, answer)
        if match and match.group(1) in answer_choice_tokens:
            return match.group(1)
    return None
```

3.3.2 Implementation

To implement my own evaluation code and demonstrate how accuracies can be improved through various methods, I began making modifications to the code structure and prompting.

First, I began with the same prompt used in the ScienceQA benchmark examples, achieving the aforementioned high accuracies on the OpenAI models but low accuracy on the open-source ones. In order to further emphasize the desire for adherence to an output format, I first experimented with various re-wordings of the prompt, adding "absolutely", or "please" and using different formats like *"Answer: <single digit>", or "Choice <single digit>"*. Eventually, due to a mistake of appending the same prompt twice, I found that simply asking the model a second time to adhere to the format led to a massive increase in accuracy for the two open-source MLLMs, while making little difference to gpt-4o and 4o-mini or Qwen2.5 text-only. Qwen2-VL increased in accuracy from 18.7% to 59.7% (and invalid answers decreased from over 700 to 94), and InternVL also increased in accuracy by over 12%.

However, despite these improvements, there were still a high number of invalid answers. A closer look revealed that sveveral of the models often returned only a single character or number as their final answer, despite being prompted to return a specific format. So, to address this issue, I simply added to the very end of the regex pattern list a pattern to match any single character within the valid answer choice list. This does risk matching random letters in explanation text such as "A" and missing the actual answer, but because the pattern is last in the list the assumption is that the previous patterns will have caught any intended answers before this. Surprisingly, this small change led to a massive decrease in the number of invalid answers for QwenVL, dropping from 94 to 11. However, InternVL saw minimal change, actually producing one additional invalid answer.

As seen in figure 2, these simple arbitrary changes to prompting and output matching were able to significantly increase benchmark performance.

4 Results

4.1 Improvements to Benchmark Evaluation

In the process of evaluating my system, I used various methods to improve the quality and efficiency of my code. Primarily, as opposed to the way typical evaluation code loops iteratively through each benchmark question, I use Python's Multiprocessing library to parallelize model calls, greatly speeding up the execution of evaluation code. However, this proved complicated to implement for some open-source models which had issues accessing inputs being stored on separate devices.

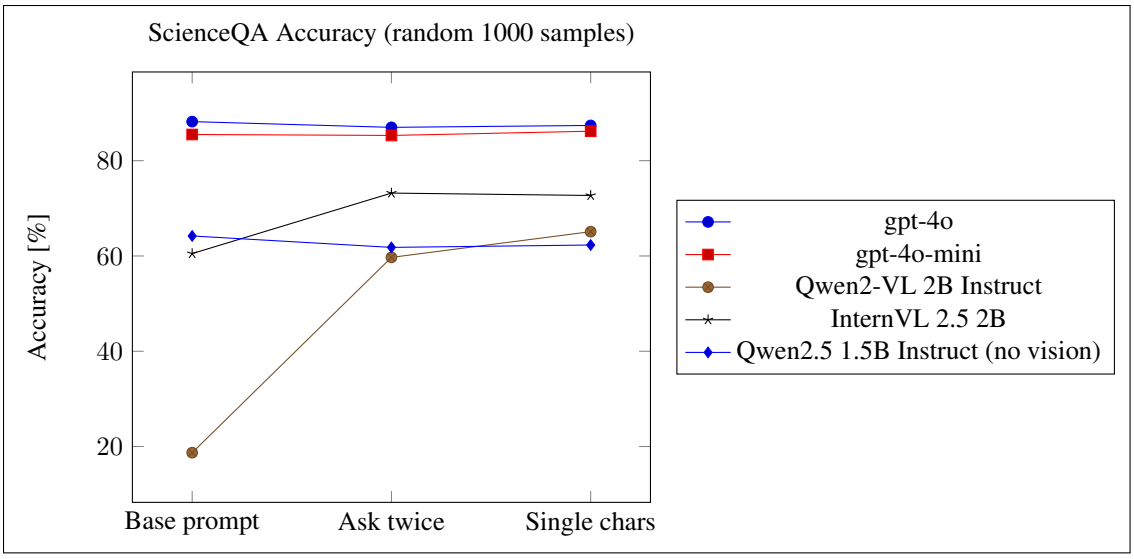


Figure 2: Sample figure caption.

Furthermore, I have structured my code to be compatible with multiple benchmarks (currently 2 but able to be expanded to more), keeping an as consistent as possible prompting and answer processing structure as possible for a more robust comparison. Currently, because each benchmark is evaluated using arbitrarily different code that is designed to give the highest possible accuracy for that dataset, metrics are questionable ways to actually compare the capability of models.

4.2 Experimental Results

Below are a collection of all of the benchmark evaluation scores I achieved throughout my testing. Significant improvements are bolded. Overall, as explained by my earlier investigation into the inconsistencies of MLLM benchmarking in general, the results here are not quite conclusive. However, as all of the following results were achieved using the same exact prompts (when possible; some formatting was required just for different model configurations on huggingface), the relative improvements are indicative that the methods have some improvement. In general, majority voting techniques seem to have marginally better results compared to the baseline, while the auto-subtask mode saw little improvement.

It is important to note here that due to GPU memory limit issues, the auto-subtask experiment was run with only two agents in the roster (Qwen2-VL and Qwen2.5 text). Furthermore, I was not able to use an open-source model as the commander, due to inability to consistently get structured output in order to coordinate the other agents. So, a major limitation here is that the auto-subtask results may largely be determined by the performance of gpt-4o itself, and not the actual structure of multiple agents. Results can be seen in the tables below 1,2,3

5 Conclusions

In conclusion, I believe the primary contribution of my work is not the initially intended novel multi-agent framework, rather a critical investigation and discussion about the limitations of current benchmark standards. As demonstrated in my results and code, the results gathered through evaluation methods such as what appears to be currently used are extremely unstable. It brings to question whether the great benchmarks that new models are supported by are more reflective of the ability to engineer evaluation prompts rather than the actual intrinsic ability of the model. Not only do these evaluations make arbitrary choices about prompting, but there are inconsistencies with whether examples are provided to the model during testing (although occasionally indicated, e.g. "0-shot results").

ScienceQA (test split 1000 random samples)	
Model	**Score (%)**
GPT-4o	87.4
GPT-4o-mini	86.1
Qwen2-VL 2B Instruct	65.1
InternVL 2.5 2B	72.7
Qwen2.5 1.5B Instruct (no vision)	62.3
Auto-subtask (gpt-4o + internvl + qwen2.5)	86.8
GPT-4o (majority vote)	86 (-1.4)
GPT-4o-mini (majority vote)	86.4 (+0.3)
Qwen2-VL 2B Instruct (majority vote)	59.7 **(+5.4)**
InternVL 2.5 2B (majority vote)	74.9 **(+2.2)**
Qwen2.5 (majority vote)	64.1 **(+1.8)**

Table 1: Accuracy of baseline models, majority voting configurations, and auto-subtask setups on ScienceQA. A random 1000 sample subset is chosen with the same seed for all models for consideration of evaluation time.

MMMU Physics (validation split)	
Model	**Score (%)**
GPT-4o	72.41
GPT-4o-mini	58.62
Qwen2-VL 2B Instruct	17.24
InternVL 2.5 2B	41.38
Qwen2.5 1.5B Instruct (no vision)	20.69
GPT-4o (majority vote)	65.52 (-6.89)
GPT-4o-mini (majority vote)	65.52 **(+8.9)**
InternVL 2.5 2B (majority vote)	27.59 (-13.79)
Auto-subtask (gpt-4o,InternVL,Qwen2.5)	48.28

Table 2: Accuracy of models on MMMU Physics validation split. This subset contains 29 college-level physics problems with images.

Furthermore, I demonstrate that using MLLMs as coordinators to dynamically assign subtasks to other agents could be a viable approach to agentic reasoning, although further experimentation is needed to improve and test the performance of the system. Additionally, I believe that the evaluation framework I develop could be one step closer towards having more reliable benchmarking standards.

Admittedly, it is inherently very difficult to standardized language models, which themselves are not deterministic. However, I strongly believe that there is a need for the development of more consistent practices for testing them, in addition for expectations of transparency about techniques used to improve benchmark performance when using them as evidence for model ability.

5.1 Future Work

To further improve benchmarking and multimodal reasoning, I intend to continue to refine the already public MMO framework and evaluation system. With more resources and time, I believe this system could be a very useful tool for accurately comparing top models. Furthermore, the MMO multi-agent framework has many possibilities in terms of organization strategies, and I intend to investigate the properties and performance benefits of these approaches as well.

MMMU Computer Science (validation split)	
Model	**Score (%)**
GPT-4o	59.26
GPT-4o-mini	40.74
Qwen2-VL 2B Instruct	-
InternVL 2.5 2B	33.33
Qwen2.5 1.5B Instruct (no vision)	25.93
GPT-4o (majority vote)	59.26 (-)
GPT-4o-mini (majority vote)	48.15 **(+7.41)**
Auto-subtask (gpt-4o,InternVL,Qwen2.5)	62.96 **(+3.7)**

Table 3: Accuracy of models on MMMU Computer Science. Auto-subtask achieves a higher score compared to gpt-4o.

References

[1] 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. URL https://arxiv.org/abs/2308.12966.

[2] Pei Chen, Boran Han, and Shuai Zhang. Comm: Collaborative multi-agent, multi-reasoning-path prompting for complex problem solving, 2024. URL https://arxiv.org/abs/2404.17729.

[3] Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, Bin Li, Ping Luo, Tong Lu, Yu Qiao, and Jifeng Dai. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks, 2024. URL https://arxiv.org/abs/2312.14238.

[4] Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate, 2023. URL https://arxiv.org/abs/2305.14325.

[5] Sirui Hong, Mingchen Zhuge, Jiaqi 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, 2024. URL https://arxiv.org/abs/2308.00352.

[6] Jian Li, Weiheng Lu, Hao Fei, Meng Luo, Ming Dai, Min Xia, Yizhang Jin, Zhenye Gan, Ding Qi, Chaoyou Fu, Ying Tai, Wankou Yang, Yabiao Wang, and Chengjie Wang. A survey on benchmarks of multimodal large language models, 2024. URL https://arxiv.org/abs/2408.08632.

[7] Pan Lu, Swaroop Mishra, Tony Xia, Liang Qiu, Kai-Wei Chang, Song-Chun Zhu, Oyvind Tafjord, Peter Clark, and Ashwin Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering, 2022. URL https://arxiv.org/abs/2209.09513.

[8] OpenAI. Learning to reason with llms, 2024. URL https://openai.com/index/learning-to-reason-with-llms/.

[9] Yiqi Wang, Wentao Chen, Xiaotian Han, Xudong Lin, Haiteng Zhao, Yongfei Liu, Bohan Zhai, Jianbo Yuan, Quanzeng You, and Hongxia Yang. Exploring the reasoning abilities of multimodal large language models (mllms): A comprehensive survey on emerging trends in multimodal reasoning, 2024. URL https://arxiv.org/abs/2401.06805.

[10] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

[11] Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, Cong Wei, Botao Yu, Ruibin Yuan, Renliang Sun, Ming Yin, Boyuan Zheng, Zhenzhu Yang, Yibo Liu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi, 2024. URL https://arxiv.org/abs/2311.16502.