

Benchmark Pipeline Development

Hsieh Yu Hsuan
School of Computing
National University of Singapore

CONTENTS

I	Task Description	1
II	Workflow	1
III	Dataset Description	1
IV	Model API: OpenAI API	2
IV-A	Prompt Design	2
IV-B	Image Input Format	2
V	Evaluation Results	2
VI	Unit Testing	3
VII	AI Usage In The Work	3
	References	3

LIST OF FIGURES

1	Project workflow	1
2	Results of OpenAI's o4-mini model	2
3	Unit test coverage report	3

LIST OF TABLES

I	Accuracy of different OpenAI models on 20 computer science questions from the MMMU dataset.	2
---	---	---

Benchmark Pipeline Development

I. TASK DESCRIPTION

The primary objective of this benchmark pipeline is to automatically evaluate the performance of vision-language models on multiple-choice questions (MCQs) from the MMMU dataset. The MMMU dataset contains multimodal problems that combine images with textual questions, making it a good benchmark for multimodal reasoning tasks.

The pipeline is designed to:

- Evaluate multiple OpenAI model variants (e.g., gpt-4o, gpt-4o-mini) on the MMMU dataset.
- Handle image + text + options inputs and produce standardized single-letter predictions (A, B, C, D).
- Compute accuracy scores over a chosen subject domain (e.g., Accounting, Computer_Science).
- Serve as a scalable and reusable benchmark for future model comparison.

II. WORKFLOW

Our evaluation pipeline is designed to benchmark multiple OpenAI API models efficiently on the MMMU dataset. Users interact with the pipeline through a command-line interface, where they can specify three key parameters to control the evaluation:

- **Model (Required):** The OpenAI API model to be evaluated (e.g., gpt-4o-mini, o4-mini).
- **Subject (Optional):** The subject split from the MMMU dataset to test (e.g., Accounting, Computer_Science), default Accounting.
- **Max Samples (Optional):** The number of samples to include in the evaluation, allowing users to limit or expand the benchmark size, default 10.

Once these parameters are defined, the pipeline automatically loads the specified subset of the dataset, sends the questions to the chosen model, and computes the accuracy. This design enables easy evaluation and comparison of multiple OpenAI API models across different subjects and sample sizes.

The end-to-end workflow consists of the following steps, as shown in Figure 1:

- `data_loader.py`: Loads the MMMU dataset, extracts images, options, and correct labels.
- `model_interface.py`: Sends inputs to the chosen OpenAI model and normalizes outputs.
- `evaluator.py`: Runs evaluation loop, enforces single-letter prediction, computes accuracy.
- `run_benchmark.py`: CLI entry point for easy execution and experimentation.

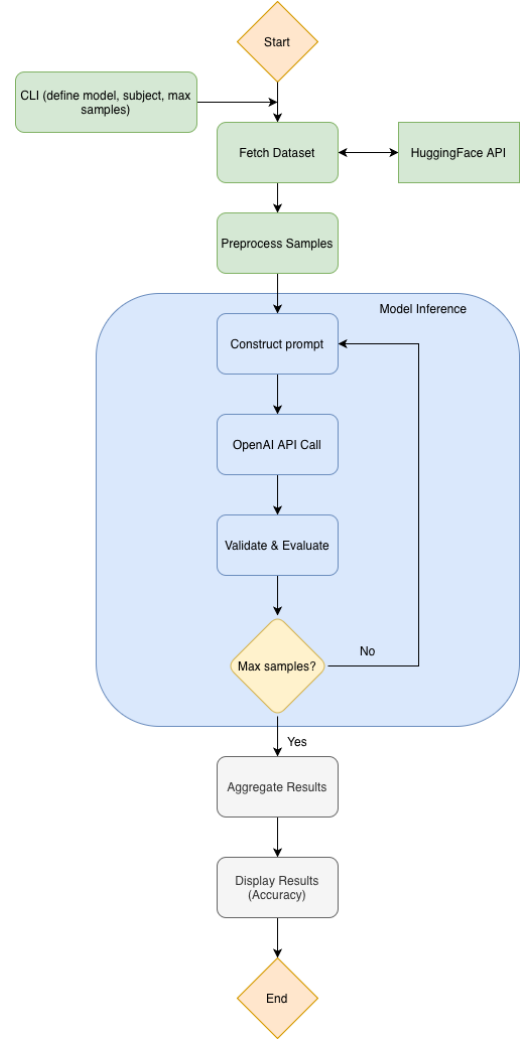


Fig. 1. Project workflow

III. DATASET DESCRIPTION

The MMMU (Massive Multimodal Multitask Understanding) dataset [1] is a comprehensive benchmark designed to evaluate the reasoning ability of multimodal models across a wide range of academic and professional domains. Unlike typical vision-language datasets that focus on object recognition or captioning, MMMU emphasizes complex reasoning tasks that require understanding both images and text simultaneously.

Each sample consists of:

- A visual input (e.g., diagram, chart, medical image, engineering schematic).
- A textual question.

- A set of multiple-choice options.
- A ground truth label indicating the correct option (typically A, B, C, D).

MMMU covers 30 subjects and 183 subfields, which makes the dataset an effective benchmark for generalist multimodal reasoning models. To ensure consistent and fair evaluation, we only consider questions that include multiple-choice options, samples without options are filtered out from the benchmark.

IV. MODEL API: OPENAI API

The OpenAI API provides access to a family of advanced multimodal language models capable of processing both text and images. For this benchmark pipeline, the API serves as the inference engine that receives the formatted question (text + image) and returns the model's predicted answer.

The API format is described in [2]. An example API call snippet is shown as following:

Listing 1. Example OpenAI API Call

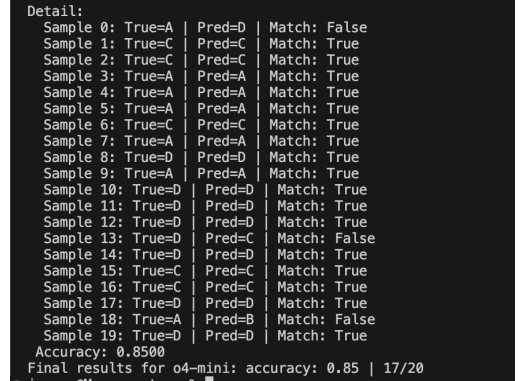
```
response = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
        {
            "role": "user", "content": [
                {
                    "type": "text", "text":
                        prompt_text,
                },
                {
                    "type": "image_url", "image_url":
                        image_b64
                }
            ]
        }
    ]
)
answer = response.choices[0].message["content"]
answer.strip()
```

A. Prompt Design

The design of the prompt plays a crucial role in determining the quality and reliability of model responses. Well-structured prompts help reduce ambiguity and guide the model toward producing concise and correct outputs. For this project, we used the following prompt to ensure that the model outputs only one valid option for each multiple-choice question:

```
prompt = (
    "You are answering a multiple-choice\n"
    "question.\n"
    "Return EXACTLY ONE letter from [A|B|C|D|E\n"
    "|F]. No other text.\n\n"
    f"Question: {question}\n\nOptions:\n{"\n"
    "options_text}\n"
    "Answer (one letter only):"
)
```

This prompt explicitly instructs the model to return a single letter corresponding to the selected answer, which minimizes the need for post-processing and improves the consistency of the outputs.



Sample	True	Pred	Match
Sample 0:	True=A	Pred=D	Match: False
Sample 1:	True=C	Pred=C	Match: True
Sample 2:	True=C	Pred=C	Match: True
Sample 3:	True=A	Pred=A	Match: True
Sample 4:	True=A	Pred=A	Match: True
Sample 5:	True=A	Pred=A	Match: True
Sample 6:	True=C	Pred=C	Match: True
Sample 7:	True=A	Pred=A	Match: True
Sample 8:	True=D	Pred=D	Match: True
Sample 9:	True=A	Pred=A	Match: True
Sample 10:	True=D	Pred=D	Match: True
Sample 11:	True=D	Pred=D	Match: True
Sample 12:	True=D	Pred=D	Match: True
Sample 13:	True=D	Pred=C	Match: False
Sample 14:	True=D	Pred=D	Match: True
Sample 15:	True=C	Pred=C	Match: True
Sample 16:	True=C	Pred=C	Match: True
Sample 17:	True=D	Pred=D	Match: True
Sample 18:	True=A	Pred=B	Match: False
Sample 19:	True=D	Pred=D	Match: True

Accuracy: 0.8500
Final results for o4-mini: accuracy: 0.85 | 17/20

Fig. 2. Results of OpenAI's o4-mini model

B. Image Input Format

In addition to textual prompts, questions in the MMMU dataset involve visual reasoning. For these cases, images are provided to the OpenAI API as part of the input payload. To provide a local image to the OpenAI API without uploading it to an external server, we convert the image into a base64-encoded data URL by first saving the image as a JPEG using a BytesIO buffer, ensuring it is in RGB format and with sufficient quality. The resulting byte stream is then base64-encoded and prefixed with the `data : image/jpeg;base64,` header.

V. EVALUATION RESULTS

We evaluated several OpenAI models on the **MMMU dataset**, focusing on the **computer science subject** with a subset of 20 samples. Our benchmark pipeline is capable of evaluating multiple OpenAI API models in a consistent and automated manner. The benchmark was conducted using the following command:

```
python3 ./A2/run_benchmark.py --model <
model_name> --max_samples 20 --subject
Computer_Science
```

The results are summarized in Table I, a snapshot of the execution terminal output is shown in Figure 2.

Model	Correct / Total	Accuracy
gpt-4o-mini	5 / 20	0.25
gpt-4o	12 / 20	0.60
o4-mini	17 / 20	0.85

TABLE I. ACCURACY OF DIFFERENT OPENAI MODELS ON 20 COMPUTER SCIENCE QUESTIONS FROM THE MMMU DATASET.

From the evaluation, we observe that o4-mini achieved the highest accuracy of 0.85, significantly outperforming gpt-4o and gpt-4o-mini. This suggests that the o4-mini model is better suited for answering computer science multiple-choice questions in this dataset, while the smaller or less advanced variants show lower performance.

Name	Stmts	Miss	Cover	Missing
data_loader.py	22	0	100%	
evaluator.py	52	7	87%	14-15, 47-52
model_interface.py	27	12	56%	26-58
run_benchmark.py	16	1	94%	24
tests/test_data_loader.py	154	1	99%	281
tests/test_evaluator.py	100	1	99%	201
tests/test_model_interface.py	14	1	93%	20
tests/test_run_benchmark.py	27	1	96%	35
tests/test_utils.py	48	0	100%	
utils/metrics.py	3	0	100%	
TOTAL	463	24	95%	

Fig. 3. Unit test coverage report

VI. UNIT TESTING

To ensure code reliability and maintainability, we have implemented unit tests for all functions in the codebase. A significant portion of these tests were generated with the assistance of AI tools, including Claude and GitHub Copilot, which helped automate test creation and reduce manual effort. Our testing framework covers a wide range of input cases and edge scenarios, achieving an overall test coverage of approximately 95%, as shown in Figure 3. The high coverage ensures that the implementation behaves as expected and facilitates easier future modifications and extensions of the code.

VII. AI USAGE IN THE WORK

AI tools were actively leveraged throughout the development process to enhance productivity, generate ideas, and streamline implementation. Specifically:

- **Idea Generation and Discussion:** ChatGPT [3] was used to brainstorm and discuss implementation strategies, helping to refine the workflow and identify potential challenges early in the process.
- **Code Development and Debugging:** GitHub Copilot [4], integrated in VSCode, and Claude [5] were used to generate initial code skeletons based on the discussed workflow, as well as to assist in debugging and iterating on code efficiently.
- **Document Preparation and LaTeX Support:** AI features in Overleaf [6] were utilized to debug LaTeX syntax issues, format equations, and generate tables, reducing manual effort and ensuring correctness.
- **Report Refinement:** ChatGPT was also used to improve the clarity, structure, and readability of this report, providing suggestions for phrasing and organization.
- **Additional Assistance:** AI tools were also consulted for quick clarification of technical concepts, best practices in coding, and suggestions for optimizing algorithms, which helped improve both the quality and speed of development.

REFERENCES

- [1] 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 Wenhao Chen. *MMMU: A Massive Multi-discipline Multimodal Understanding and Reasoning Benchmark for Expert AGI*. Proceedings of CVPR, 2024.
- [2] OpenAI. *OpenAI API Reference*. Available at: <https://platform.openai.com/docs>. Accessed: October 20, 2025.

- [3] OpenAI. *ChatGPT (GPT-4)* [Large language model]. OpenAI, 2025. Available at: <https://chat.openai.com>.
- [4] GitHub. *GitHub Copilot* [AI-powered code completion]. GitHub, 2025. Available at: <https://github.com/features/copilot>.
- [5] Anthropic. *Claude* [Large language model]. Anthropic, 2025. Available at: <https://www.anthropic.com/claude>.
- [6] Overleaf. *Overleaf AI features* [AI-assisted writing and LaTeX support]. Overleaf, 2025. Available at: https://www.overleaf.com/learn/how-to/AI_features.