Evaluating Multimodal Language Models on Visual Puzzles and Brainteasers

Nikhilesh Joshi (230700), Kamatam Akshay Reddy (230517), Kishore Senthilkumar (230566)

April 2025

1 Abstract

With the rise of powerful multimodal language models that can understand both text and images, it's become possible to test their reasoning skills in creative ways. In this project, we put these models to the test using a collection of visual puzzles and brainteasers we gathered manually from across the internet—things like pattern games, logic-based challenges, and optical illusions. We curated, cleaned, and organized the dataset ourselves, then evaluated how well models like Gemini, DeepSeek, and Perplexity performed on these tasks. By analyzing their answers, explanations, and consistency, we gained insights into how effectively current AI systems handle abstract visual reasoning. Our results shed light on both the strengths and limitations of these models when faced with puzzles designed to challenge human thinking.

2 Introduction

In recent years, artificial intelligence has taken significant strides with the emergence of Large Language Models (LLMs), and more recently, Multimodal Language Models (MLLMs) — systems capable of processing and reasoning over both textual and visual data. These models, including industry-leading examples like GPT-4o, Gemini, Claude, and LLaMA, are transforming the landscape of machine intelligence. With this evolution comes a fundamental question: Can these models reason visually as effectively as humans, or are they merely leveraging pattern recognition and memorization?

Traditional benchmarks often blur the line between true reasoning and shortcut-based inference. As a result, there is a growing need for rigorous evaluation methods that focus on core reasoning abilities, especially in visually grounded contexts. This project takes a novel approach by evaluating the visual reasoning capabilities of MLLMs through a curated set of brainteasers and visual puzzles — tasks that inherently require a blend of logical deduction, pattern recognition, and perceptual acuity.

Our primary goal is to assess how well state-of-the-art MLLMs can interpret and solve such challenges. To do this, we constructed a diverse dataset sourced from platforms like JagranJosh, known for realistic and human-interaction-based puzzles, and Pinterest, which offers a rich collection of logical and mathematical visual problems. These puzzles were organized into distinct categories such as "Find the Mistake," "Counting Figures," "Maze," and others to facilitate a focused analysis across different cognitive tasks.

To evaluate performance, we employed APIs for MLLMs like Gemini and DeepSeek, designing Python scripts to automate the image input and capture the models' reasoning responses in structured formats. To address the potential of memorization, we supplemented our dataset with a set of novel, hand-crafted puzzles that had never appeared online, ensuring a robust test of the models' generalization and reasoning capabilities.

This combined effort allows us to move beyond surface-level evaluations and explore whether current models exhibit genuine visual reasoning — or if they simply excel at reproducing learned patterns.

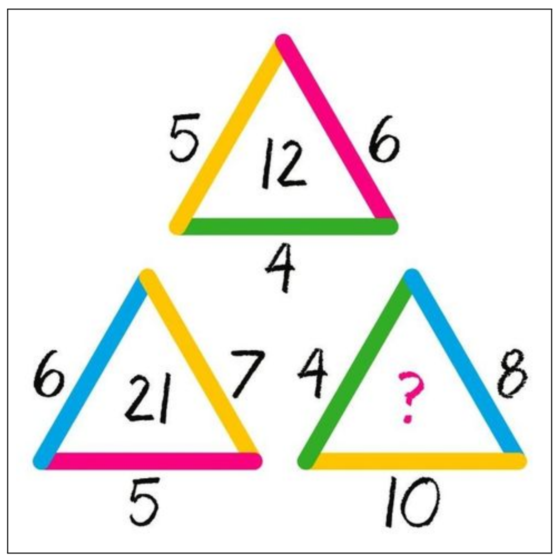


Figure 1: Puzzles that require establishing relationship between numbers in a non mathematical way

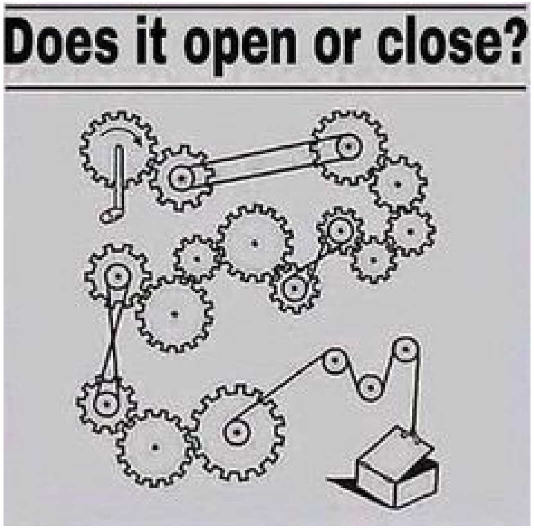


Figure 2: Puzzles that involved spatial reasoning

3 Preparing Datasets

To evaluate the visual reasoning abilities of multimodal language models, we curated a diverse and challenging dataset composed of brainteasers and visual puzzles. A significant portion of our dataset was constructed using images sourced from websites like JagranJosh and Pinterest. JagranJosh was chosen for its realistic, everyday puzzles that simulate human-like visual reasoning, requiring keen observation and logical deduction. These challenges closely mirror situations a human might encounter in daily life. Pinterest, on the other hand, provided a wide range of abstract and logic-driven puzzles such as pattern completions, spatial reasoning tasks, and mathematical riddles. To ensure comprehensive coverage, we manually categorized these puzzles into types like "Counting Figures," "Calculate the Answer," and "Riddles" allowing for targeted evaluation across different reasoning domains.

Figure 3: Puzzles with trick questions

In addition to web-sourced puzzles, we incorporated selected datasets from Hugging Face, specifically aiming for collections that the LLMs under evaluation were unlikely to have encountered during pretraining. The motivation behind this was to ensure that model performance wasn't the result of simple memorization, but rather indicative of genuine reasoning ability. By testing the models on novel, lesser-known puzzles, we were able to more rigorously assess whether their responses reflected true intelligence or they were simply memorised by the LLMs as they were easily available on the internet. This step was crucial in distinguishing between memorization and meaningful visual understanding—core aspects of what we define as intelligent behavior in artificial systems.

Figure 4: Puzzles which required some knowledge of the real world

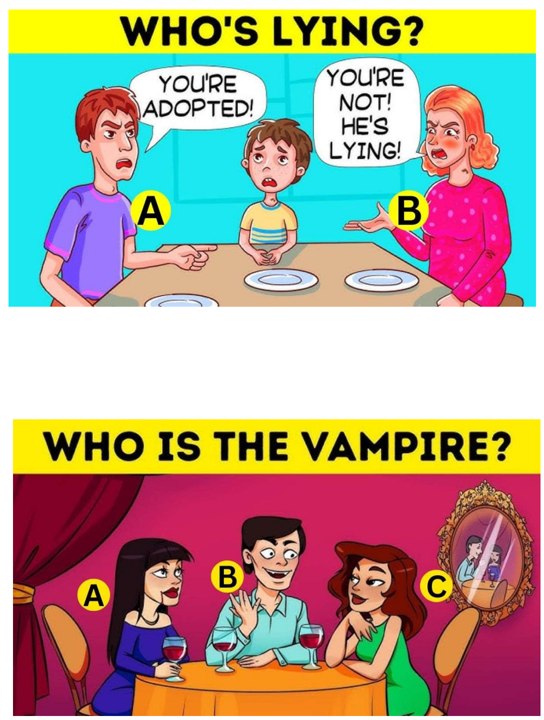


Figure 5: *Puzzles that involved finding clues or hints within the images*

4 Evaluation of the LLMs

We conducted an evaluation of the responses that both the LLMs, Deepseek and Gemini, and some of the responses received are presented here. Before evaluating, we classified images into various classes and also categorized them as hard and easy based on the complexity of the puzzle. Then, we evaluated LLMs on each of these classes independently.

For evaluating Gemini, we accessed the API key of Gemini and wrote a python script which takes input as a folder of images and outputs a csv which contains the answer and reasoning given by the LLM.

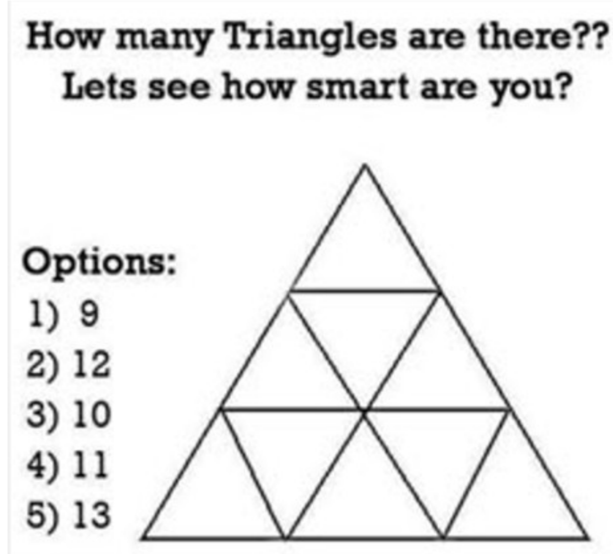
To evaluate Deepseek, we utilized the model deepseek-ai/deepseek-vl-1.3b-chat hosted on Hugging Face. After installing the required dependencies, we provided an appropriate system prompt, based on which the model generated responses along with its reasoning.

(a) Question involving observations

```
iq-test-brain-teasers-with-answers_jpg.jpg|
"The robot is Nurse B.  Here's why:

1. **The Neck:** Look closely at the backs of the
nurses in the background.  Nurse A's counterpart has
a perfectly normal neck. Nurse B's counterpart,
however, has a small, circular object on her neck
that looks suspiciously like a charging port or some
other technological component not usually seen on humans.

2. **The Trays:** While seemingly minor, the
difference in the food the nurses are carrying is
also a subtle clue. Nurse A has a meal that is
relatively standard fare, while Nurse B's tray
is a light pink and contains what seem to be pills
or some type of medication.  This suggests a less
personalized approach, typical of a robot programmed
 to carry and distribute things.

3. **Overall Impression:** Nurse B's slightly less
expressive face and less natural looking hair/facial
features when compared to nurse A also contribute
to the overall idea of her being artificial.  It's the combined
details that point to Nurse B.

Therefore, Nurse B's features point towards her being a robot in
comparison to Nurse A."| correct.
```

(b) Response Received

Figure 6: LLM made the correct observation



(a) Question involving counting

```
c572f9ea4f9bbcc15a22796d81a6af13.jpg,"Here's how to solve
this triangle counting puzzle:

**1. Identify the smallest triangles:**

Start by counting the smallest triangles.  There are nine
of these tiny triangles that make up the larger figure.

**2. Count the medium-sized triangles:**

Next, look for triangles composed of four of the smallest
triangles. There are three of these medium-sized triangles.

**3. Count the large triangle:**

Finally, count the largest triangle that encompasses the
entire figure. There is only one.

**4. Add them up:**

Add the number of small, medium, and large triangles
together: 9 + 3 + 1 = 13.

**Therefore, the answer is 5) 13**", correct
```

(b) Response Received

Figure 7: Evaluating responses received from LLMs

(a) Question involving observations

```
95475a9361ff34ab3a250e260b7eee0a.jpg,"Here's how to
solve the animal puzzle:

1. **The Obvious:**  Immediately, we see a large
elephant. That's one animal.

2. **Hidden within the Elephant:** Look closely
at the elephant's silhouette.
Inside the elephant's shape, you can see the
outline of a donkey. That's two animals.

3. **More Hidden Animals:** Within the donkey's
shape, there's a smaller dog.
That's three animals.  And finally, a tiny
dog appears within the small dog,
making a total of four animals.


Therefore, the answer is **four** animals.",
wrong: mouse and mosquito and more missing.
```

(b) Response Received

Figure 8: LLM was not able to make all the observations

After receiving the responses for all the puzzles, we manually checked the responses and the observations are tabulated below.

Model	Accuracy (in %)	Imgs
Deepseek-VL-13.b	51.12	2458
Gemini-1.5 Flash	56.56	

Table 1: For logic puzzles

Model	Accuracy (in %)	Imgs
Deepseek-VL-13.b	35	512
Gemini-1.5 Flash	27.6	

Table 2: For counting shapes in a figure puzzles

Model	Accuracy (in %)	Imgs
Deepseek-VL-13.b	74.34	947
Gemini-1.5 Flash	65.4	

Table 3: For riddles

Model	Accuracy (in %)	Imgs
Deepseek-VL-13.b	48.79	836
Gemini-1.5 Flash	58.34	

Table 4: For finding mistakes in the image puzzles

Model	Accuracy (in %)	Imgs
Deepseek-VL-13.b	67.23	2135
Gemini-1.5 Flash	55.41	

Table 5: For arithmetic puzzles

Model	Accuracy (in %)	Imgs
Deepseek-VL-13.b	43.23	678
Gemini-1.5 Flash	47.1	

Table 6: For puzzles requiring keen observations

Model	Accuracy (in %)	Imgs
Deepseek-VL-13.b	21.09	423
Gemini-1.5 Flash	23.45	

Table 7: For finding the next element in a sequence puzzles

However, apart from differentiating puzzles on the basis of types, we also valuated models on images that were easily available on the internet and those that the models probably did not encounter while pretraining. Based on this differentiation, the observations are as follows.

Model	Easy	Hard	Imgs
Deepseek-VL-13.b	68.19	37.28	7989
Gemini-1.5 Flash	59.63	29.4	

Table 8: Performance of models for different puzzles. Here Easy means the puzzles that were easily available for the models to train upon and Hard refers to the puzzles that are not yet encountered by the models.

5 Conclusions

Despite how convincing the responses from Multimodal Language Models might appear, a deeper analysis reveals a more limited form of capability. These models excel at recalling patterns, matching language, and generating outputs based on statistical probabilities — all grounded in the vast datasets they have been trained on. What may look like intelligence is often the result of exposure to similar examples during training, not an indication of genuine reasoning or understanding. In many cases, their answers are closer to regurgitation than real insight.

True intelligence involves the ability to reason through unfamiliar problems, draw abstract connections, and demonstrate creativity or critical thinking — qualities that current AI systems largely lack. As long as these models rely on pre-learned data to produce responses, they will continue to mimic intelligence rather than possess it. Until AI can independently interpret and adapt to new situations without relying on what it has seen before, it remains clear that these systems are not truly intelligent — just remarkably good at memorizing.

6 Learning from the Project

Throughout the course of this project, we uncovered several key insights related to the integration of image inputs with large language models (LLMs) and the challenges inherent in visual reasoning. The following are the primary lessons learnt:

- **Importance of Preprocessing in Image-to-Text Pipelines:** One of the most critical findings was the impact of preprocessing on model performance. Fine-tuning elements such as image resolution, input quality, and batch size significantly improved the accuracy of image-to-text conversions. This highlighted that preprocessing is not merely a preparatory step, but a decisive factor in the effectiveness of the pipeline.
- **Inconsistency Across LLM Interpretations:** We observed a considerable degree of variation in how different LLMs interpreted the same visual inputs. These differences stem from variations in model architecture, training datasets, and internal biases. This finding emphasizes the subjective and model-specific nature of AI-driven reasoning and the need for consistent benchmarking methods.
- **Pinpointing Reasoning Limitations in Visual Inputs:** The project enabled us to identify specific image categories where LLMs exhibited reasoning failures. In many such cases, the models were unable to capture essential visual features, leading to incorrect or incomplete responses. Recognizing these limitations is essential for guiding future improvements in visual understanding and reasoning capabilities.

7 Future Scope following this Project

The findings from this project pave the way for several promising directions to further enhance the integration of image understanding with large language models (LLMs). The following points outline the key areas for future work:

1. **Expand Dataset Diversity:** Incorporate a wider range of datasets, especially those that include multilingual brain teasers and puzzles. This would allow the evaluation of LLM performance in cross-linguistic reasoning scenarios and improve their generalization across languages.
2. **Automated Answer Verification:** Develop a backend system to automatically compare LLM responses with ground-truth answers (where available) using methods such as keyword matching or fuzzy logic. For more subjective, open-ended tasks, embedding-based similarity techniques (e.g., Sentence-BERT) can be used to compare LLM-generated explanations with reference reasoning.
3. **Quantitative Evaluation Metrics:** Create structured metrics to assess response quality based on dimensions like accuracy, clarity, and logical coherence. This could involve a detailed rubric or the use of self-evaluation prompts by the LLM (e.g., "Rate the above explanation from 1 to 5").
4. **Prompt Engineering for Better Accuracy:** Investigate the impact of different prompt styles to improve reasoning quality. Examples include:
   - Chain-of-thought prompting to guide the model through multi-step reasoning.
   - Role-based prompting, such as instructing the LLM to behave as a domain expert (e.g., "Act like a puzzle master").

   Study how even small variations in prompts can influence the accuracy and depth of reasoning in model responses.

Appendix

Extracting images from JagranJosh

The following Python code demonstrates how we collected a variety of images from JagranJosh. We followed a recursive approach, in the sense that we started with a website and extracted the puzzle image that was available on the website and saved it. After this, we crawled to the end of the webpage using our script and navigated to a bunch of webpages that contained certain keywords and followed the same approach with each of them

```
import requests
from bs4 import BeautifulSoup
import csv
import time

def crawl_and_save(url, csv_writer, visited_urls):
    if url in visited_urls:
        return

    visited_urls.add(url)
    try:
        response = requests.get(url, headers={'User-Agent': 'Mozilla/5.0'})
        soup = BeautifulSoup(response.content, 'html.parser')

        links = soup.find_all('a', href=True)
        puzzle_links = [link['href'] for link in links if 'jagranjosh.com' in link['href'
            ] and
                        'general-knowledge' in link['href'].lower() and ('brain-teaser'
                            in
                        link['href'].lower() or 'optical-illusion' in link['href'].lower
                            () or
                        'iq-test' in link['href'].lower() or  'spot' in link['href'].
                            lower() or
                        'word' in link['href'].lower() or 'picture-puzzle' in link['href'
                            ].lower()
                        or 'math-puzzle' in link['href'].lower() or 'find' in link['href'
                            ].lower() or
                        x'visual' in link['href'].lower())]

        for link in puzzle_links:
            if link not in visited_urls:
                csv_writer.writerow([link])
                print(f"Added: {link}")
                crawl_and_save(link, csv_writer, visited_urls)

        time.sleep(1)  # Add a delay to be respectful to the server
    except Exception as e:
        print(f"Error crawling {url}: {str(e)}")

def main():
    start_url = "https://www.jagranjosh.com/general-knowledge/brain-teaser-find-whos-not-
       rich-in-6-seconds-1731416204-1"
    visited_urls = set()

    with open('reasoning_puzzle_links.csv', 'w', newline='', encoding='utf-8') as csvfile
        :
        csv_writer = csv.writer(csvfile)
        csv_writer.writerow(['URL'])  # Write header
        crawl_and_save(start_url, csv_writer, visited_urls)

if __name__ == "__main__":
    main()
```

Extracting images from Pinterest

The following Python code shows how we extracted images from Pinterest. Unlike JagranJosh, Pinterest presented a new challenge for us. All Pinterest websites were so long that the browser uses something called Lazy Scrolling for displaying images. This means that the browser does not load the section of images that is at the bottom till the time we do not manually navigate to the bottom. Thus, we had to implement a lazy scroller in our code that automates the scrolling part and loads images and then saves them.

```python
from selenium import webdriver
from bs4 import BeautifulSoup
import requests
import os
import time
import uuid

# Constants
PINTEREST_URL = "https://in.pinterest.com/gamespicnic/visual-brain-teasers-and-puzzles/"
SLEEP_TIMER = 2
FOLDER_NAME = "final_data_pinterest"

# Setup folder to save images
os.makedirs(FOLDER_NAME, exist_ok=True)

def setup_driver():
    """Initialize and return a Selenium Chrome WebDriver."""
    options = webdriver.ChromeOptions()
    options.add_experimental_option("excludeSwitches", ["enable-logging"])
    return webdriver.Chrome(options=options)

def scroll_to_bottom(driver, sleep_time=SLEEP_TIMER):
    """Scrolls to the bottom of a Pinterest board page to load all images."""
    last_height = driver.execute_script("return document.body.scrollHeight")

    while True:
        driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
        time.sleep(sleep_time)
        new_height = driver.execute_script("return document.body.scrollHeight")
        if new_height == last_height:
            print("Reached the bottom of the page.")
            break
        last_height = new_height

def extract_image_urls(driver):
    """Extract all unique image URLs from the loaded Pinterest page."""
    soup = BeautifulSoup(driver.page_source, 'html.parser')
    img_urls = set()

    for link in soup.find_all('img'):
        img_url = link.get('src') or link.get('data-src')
        if img_url:
            clean_url = img_url.split("?")[0]
            img_urls.add(clean_url)

    return img_urls

def download_images(image_urls, folder):
    """Download and save images to the specified folder."""
    for img_url in image_urls:
        try:
            response = requests.get(img_url, stream=True)
            if response.status_code == 200:
```

```
                filename = os.path.join(folder, f"{uuid.uuid4().hex}.jpg")
                with open(filename, 'wb') as file:
                    for chunk in response.iter_content(1024):
                        file.write(chunk)
                print(f"Downloaded: {filename}")
            else:
                print(f"Failed to download: {img_url}")
        except Exception as e:
            print(f"Error downloading {img_url}: {e}")

def scrape_pinterest_images(url):
    """Main function to scrape Pinterest images using lazy scrolling."""
    driver = setup_driver()
    driver.get(url)

    scroll_to_bottom(driver)
    image_urls = extract_image_urls(driver)
    print(f"Total images found: {len(image_urls)}")

    download_images(image_urls, FOLDER_NAME)
    driver.quit()

# Run the scraper
scrape_pinterest_images(PINTEREST_URL)
```

Evaluating the MLLMs

The following is a Python code that we have used to evaluate the LLMs Gemini 1.5 Flash and Deepseek-VL-1.3b. The code involves securing a connection with the respective APIs using the API keys that we have generated and then sending the image along with a predecided prompt to the LLMs and then storing the responses and reasonings in a CSV file that we have manually checked later.

```
import os
import pandas as pd
from PIL import Image
import google.generativeai as genai

# ========== STEP 1: Gemini API Key ==========
genai.configure(api_key="AIzaSyDNq1sts47AGQazTqFPKq2AfMXNY90tAT4")

# ========== STEP 2: Configuration ==========
image_folder = r"C:\Users\kisho\Downloads\UGP\Images\nikhilesh"
output_csv = "llm_responses_charac.csv"
model_name = "gemini-1.5-flash"
prompt = "This is a brain teaser or puzzle. Please solve it " \
"and explain your reasoning step-by-step."

# ========== STEP 3: Load Model ==========
model = genai.GenerativeModel(model_name)

# ========== STEP 4: Process Each Image ==========
results = []
image_files = sorted([f for f in os.listdir(image_folder) if
                      f.lower().endswith(('.jpg', '.jpeg', '.png'))])

for img_name in image_files:
    image_path = os.path.join(image_folder, img_name)

    try:
        img = Image.open(image_path)
        response = model.generate_content([prompt, img])
```

```
        reasoning = response.text.strip()
    except Exception as e:
        reasoning = f"Error: {e}"

    results.append({
        "image_id": img_name,
        "reasoning": reasoning,
        "inference": ""  # Empty column for now
    })

# ========== STEP 5: Save to CSV ==========
df = pd.DataFrame(results)
df.to_csv(output_csv, index=False, sep='|')

print(f"\n Saved {len(results)} responses to '{output_csv}' using '|' as delimiter.")
```

The following is the code we used to evaluate Deepseek-VL-13.b

```
import os
import torch
import csv
from transformers import AutoModelForCausalLM
from deepseek_vl.models import VLChatProcessor, MultiModalityCausalLM
from deepseek_vl.utils.io import load_pil_images

# CPU fallback
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Load model and processor
model_path = "deepseek-ai/deepseek-vl-1.3b-chat"
vl_chat_processor: VLChatProcessor = VLChatProcessor.from_pretrained(model_path)
tokenizer = vl_chat_processor.tokenizer

vl_gpt: MultiModalityCausalLM = AutoModelForCausalLM.from_pretrained(model_path,
    trust_remote_code=True)
vl_gpt = vl_gpt.to(device).eval()

# Folder with images
image_folder = r"C:\Users\kamat\Downloads\reddy\2\odd_one_out"
output_csv = r"C:\Users\kamat\Downloads\odd_one_out_output_reddy_deepseek.csv"
image_files = [os.path.join(image_folder, fname) for fname in os.listdir(image_folder) if
     fname.lower().endswith((".png", ".jpg", ".jpeg"))]

results = []

for image_path in image_files:
    print(f"Processing: {image_path}")

    conversation = [
        {
            "role": "User",
            "content": "<image_placeholder>Describe each stage of this image.",
            "images": [image_path]
        },
        {
            "role": "Assistant",
            "content": ""
        }
    ]

    try:
        pil_images = load_pil_images(conversation)
```

```
        prepare_inputs = vl_chat_processor(
            conversations=conversation,
            images=pil_images,
            force_batchify=True
        ).to(device)

        inputs_embeds = vl_gpt.prepare_inputs_embeds(**prepare_inputs)

        outputs = vl_gpt.language_model.generate(
            inputs_embeds=inputs_embeds,
            attention_mask=prepare_inputs.attention_mask,
            pad_token_id=tokenizer.eos_token_id,
            bos_token_id=tokenizer.bos_token_id,
            eos_token_id=tokenizer.eos_token_id,
            max_new_tokens=512,
            do_sample=False,
            use_cache=True
        )

        answer = tokenizer.decode(outputs[0].cpu().tolist(), skip_special_tokens=True)
        user_prompt = prepare_inputs['sft_format'][0]
        results.append([os.path.basename(image_path), user_prompt, answer.strip()])

    except Exception as e:
        print(f" Error processing {image_path}: {e}")
        results.append([os.path.basename(image_path), "ERROR", str(e)])

# Save to CSV
with open(output_csv, "w", newline="", encoding="utf-8") as f:
    writer = csv.writer(f)
    writer.writerow(["Image Filename", "Prompt", "Model Response"])
    writer.writerows(results)

print(f"\n Finished! Results saved to {output_csv}")
```