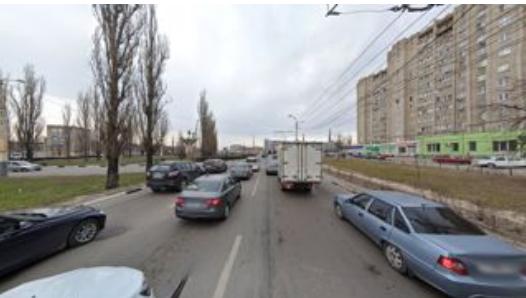


---

# GeoLocator Demo

By: Avinash Atluru, Tina Nguyen, Sarvesh Kasture, Charlie Liu

# DataSet



## 1. Street Location Images (Kaggle)

- images more applicable
  - Less data, but most systems couldn't handle more

```
▼ "root" : { 5 items
  ▼ "coordinates" : [ 2 items
    0 : float 33.7682553106586
    1 : float -84.26942443847656
  ]
  "country_name" : string "United States"
  "country_code" : string "US"
  ▼ "regions" : [ 5 items
    0 : [ ... ] 2 items
    1 : [ ... ] 2 items
    2 : [ ... ] 2 items
    3 : [ ... ] 2 items
    4 : [ ... ] 2 items
  ]
  "is_in_region" : string "True"
```

15K out of 30K sampled

- Training samples: 10500, 70.0%
- Validation samples: 2250, 15.0%
- Testing samples: 2250, 15%
- 14:3 ratio
- No descriptions
- Geotagged with coordinate locations
- Contains information about region, country, and other location info

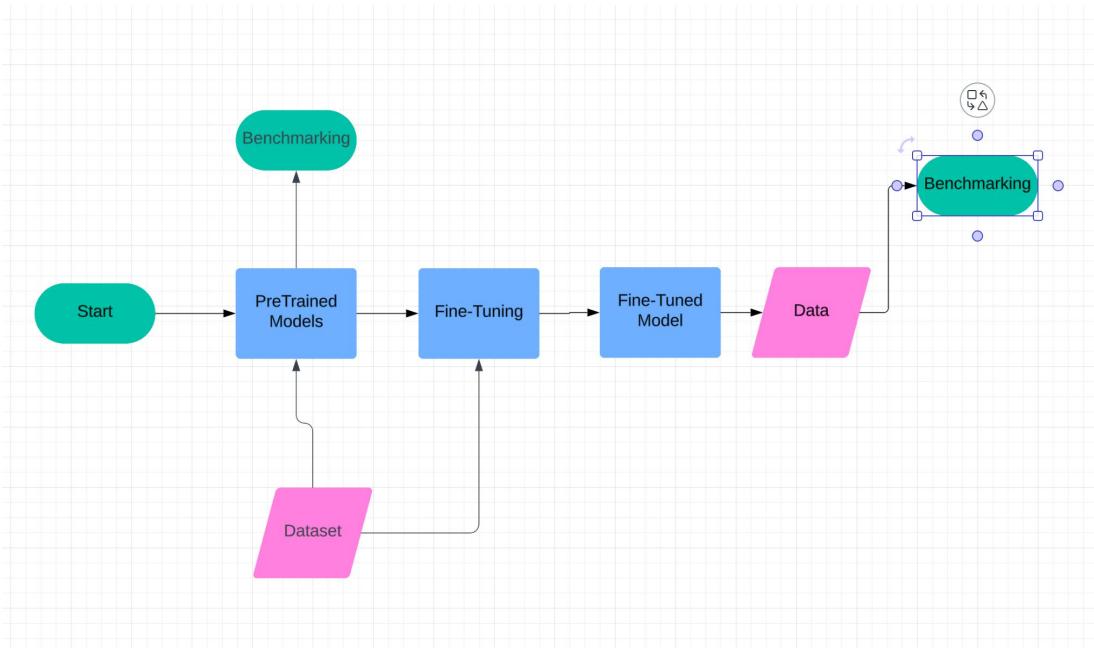
---

## Example Pictures from Test Dataset



---

# System Architecture





# System Architecture Component Breakdown

Pre-Trained Models:

- Phi 3.5: <https://huggingface.co/microsoft/Phi-3.5-vision-instruct>
- Qwen 2: <https://huggingface.co/Qwen/Qwen2-VL-2B-Instruct>
- LLaMA: <https://huggingface.co/qresearch/llama-3.1-8B-vision-378>
- Pixtral: <https://huggingface.co/mistralai/Pixtral-12B-Base-2409>

Fine Tuning: This was done using PEFT and LORA from Hugging Face:

<https://huggingface.co/docs/peft/en/index>

Fine Tuned Models (Own): Each of the 4 models were trained and saved locally to be tested again

Dataset (Extension): Kaggle Dataset mentioned in the previous slides. We took a subset and cleaned this data.

Benchmark (Own): Involved running the pre and post tuned models on a portion of the Kaggle Dataset



# Experimental Setup

Step 1: Clean the Dataset (Filter any values without location)

Step 2: Run Benchmarking of 4 Different Pre-Trained models on Dataset (Test Portion)

Step 3: Fine Tune the 4 models using the Dataset (Train Portion)

Step 4: Run Benchmarking of the 4 Different Fine-Tuned models on Dataset (Test Portion)

Step 5: Obtain Comparative Results for Pre and Post Fine Tuning (easy, medium, and hard cases)

Step 6: Obtain Comparative Results between all 4 Fine-Tuned models (easy, medium, and hard cases)

---

# Model Performance: PHI-3.5



## PHI-3.5 Vision Model Benchmarking

Percentage within 25 km: N/A

Percentage within 100 km: N/A

Percentage within 500 km: N/A

Minimum distance: N/A

Mean distance: N/A

Median distance: N/A

Standard deviation of distance: N/A

**Not Enough Data! Only Returned Coordinates for 12 Pictures while the rest returned None.**

---

## PHI-3.5 Vision Benchmarking



The model returned coordinates for this image with this response:  
To narrow down the location, the presence of a red post box is a strong indicator of a location in the UK. The style of the buildings, which seem to be modern residential apartments, could suggest a city that has undergone recent development. Without more specific landmarks, it is difficult to determine the exact city or region. However, based on the image, a rough estimate of the geolocation could be: [latitude, longitude] = [51.5074, -0.1278]

The actual coordinates are: (18.44396 47.569883, and this is actually in Hungary!

---

# Phi 3.5 Vision Benchmarking

## How are most responses?

The image shows a rural road with a clear sky, a few clouds, and a mountainous backdrop. There are no distinctive landmarks or signs that would allow for an exact geolocation. However, the landscape suggests it could be in a region with a temperate climate, possibly in Europe or North America. Without more specific information, it is not possible to determine the exact latitude and longitude. Therefore, I cannot provide an exact geolocation for this image.



The image shows a road with a pedestrian crossing, a yellow sign, and a fence on one side, with buildings and bare trees in the background. There are no distinctive landmarks or signs that would allow for an accurate geolocation. The style of the road and the fence could suggest a location in North America, but without more specific information, it is not possible to determine the exact geolocation. Therefore, I cannot provide an exact latitude and longitude for this image.

---

# Phi 3.5 Vision Fine Tuning

Statistics:

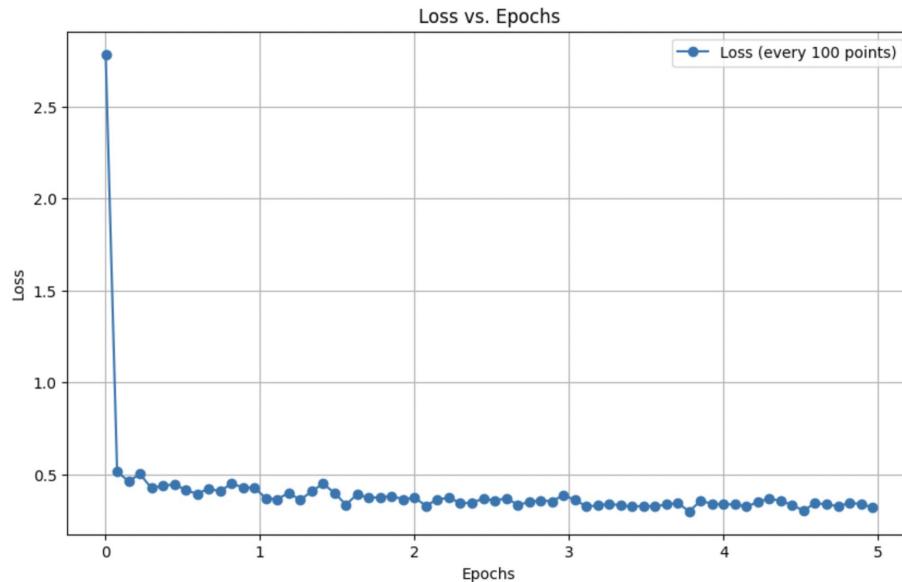
Data Processing: 3 Hours

Fine Tuning: 8 hours

Benchmarking: 2.5 Hours

Epochs: 5

Dataset Size: 15,000 images



---

# Phi 3.5 Vision Fine Tuning

## How Do Outputs Look?

[25.310001373291016, -27.992114639282227]

[50.12310791015625, -3.0112508354187012]

[29.555118560791016, -115.7911376953125]

## What was the Prompt?

Predict the geolocation of this image based on its visual cues. Think step by step, first guessing the country, then region, then the city/region, then the latitude and longitude. Please return an exact latitude and longitude in this format: [latitude, longitude]. If exact is not found, please give the latitude and longitude in this format: [latitude, longitude] for the city/region you determined."

---

# Phi 3.5 Vision Fine Tuned Statistics

This model returns coordinates for all but 13 images

Percentage within 25 km: 2.29%

Percentage within 100 km: 6.73%

Percentage within 500 km: 27.68%

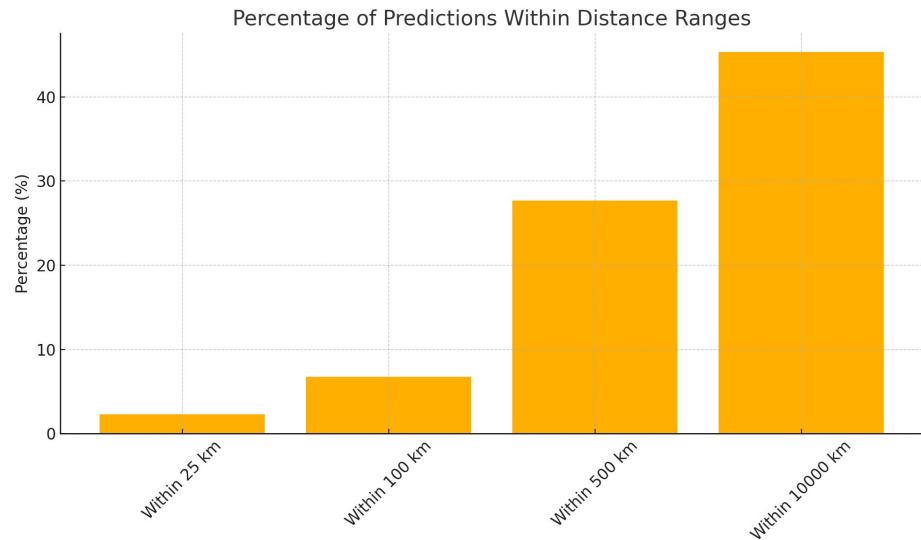
Percentage within 1000 km: 45.32%

Minimum distance: 0.88 km

Mean distance: 3692.34 km

Median distance: 1229.65 km

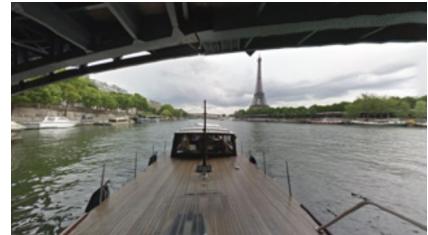
Standard deviation of distance: 4673.97 km



---

## Phi-3.5 Vision Fine Tuning Case Overview

Easy Cases: The fine tuned model got this within 2 KM in Haversine Distance, and the original model returns this: The image shows a view from a boat on a river with the Eiffel Tower in the background, which is a landmark of Paris, France. The Eiffel Tower is located in the 7th arrondissement of Paris, near the Seine River. The latitude and longitude of Paris are approximately  $48.8566^{\circ}$  N latitude and  $2.3522^{\circ}$  E longitude. Therefore, the geolocation of this image is likely around these coordinates.



Medium Cases: The image on the right shows a street, and the model was able to return these coordinates within 0.88 KM in Haversine Distance. The pretrained model was able to get the country, but it could not get an accurate location: The image shows a tree-lined street with people walking, which could be indicative of a park or a pedestrian area in a city. The architecture of the buildings in the background suggests a European style, possibly in a city with historical significance. However, without more specific landmarks or signs, it is not possible to determine the exact geolocation of this image. Therefore, I cannot provide an exact latitude and longitude.



Hard Cases: The fine tuned model was over 16000 KM off the coordinates, and the original model returned this: The image shows a rural road with a vehicle on the right side, surrounded by green fields and a few trees. There are no distinctive landmarks or signs that would allow for an accurate geolocation. The style of the road and the vegetation suggest it could be in a tropical or subtropical region. However, without more specific information, it is not possible to determine the exact geolocation of this image. Therefore, I cannot provide an exact latitude and longitude.



# Phi 3.5 Fine Tuning Code

```
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype=torch.bfloat16,
    device_map="auto",
    trust_remote_code=True,
    _attn_implementation='eager'
).to(device)

tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)
image_processor = AutoImageProcessor.from_pretrained(model_name, trust_remote_code=True)

lora_config = LoraConfig(
    r=16,
    lora_alpha=32,
    lora_dropout=0.05,
    target_modules=['q_proj', 'k_proj', 'v_proj', 'o_proj', 'fc1', 'fc2'],
    bias="none",
    task_type="CAUSAL_LM"
)
model = get_peft_model(model, lora_config)
```

```
if os.path.exists(json_path) and os.path.exists(img_path):

    with open(json_path, 'r') as f:
        metadata = json.load(f)

    caption = (
        f"Based on the visual cues in this image, identify the likely location. "
        f"First, specify the country: {metadata['country_name']}, "
        f"then the region: {metadata['regions'][0][0]}, "
        f"and finally provide approximate latitude and longitude: {metadata['coordinates']}."
    )

    entries.append({"image_path": img_path, "caption": caption})
    count += 1

images = [image]
placeholder = "<|image_1|>\n"

messages = [
    {
        "role": "user",
        "content": placeholder +
        "Based on the visual cues in this image, identify the likely location. "
        "Predict the geolocation of this image based on its visual cues. Think step by step, first guess"
    }
]

prompt = processor.tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)

inputs = processor(prompt, images=images, return_tensors="pt").to("cuda:0")

generate_ids = model.generate(
    **inputs,
    eos_token_id=processor.tokenizer.eos_token_id,
    **generation_args
)
```

---

# Model Performance: LLama-3.2

# LLama-3.1-11B-Vision

Benchmarked against 2 datasets

Issues with fine tuning:

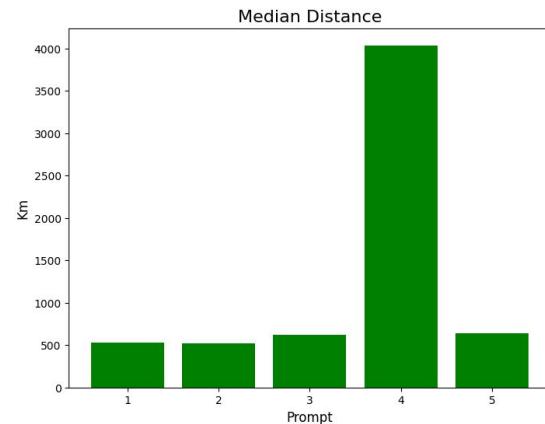
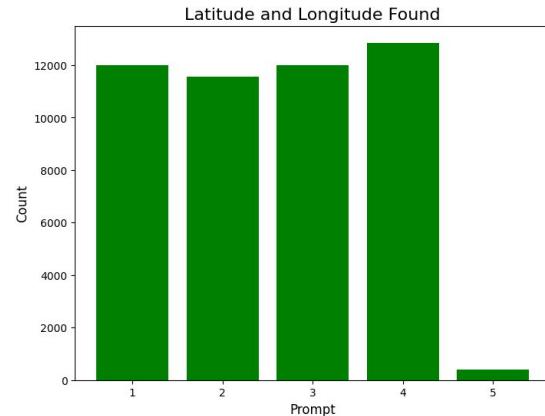
- Memory required exceeded Google Colab Pro+ allowance
- Tried multiple different optimizations
  - Gradient Checkpointing
  - LoRA Fine-Tuning
  - 8-bit Quantization

```
-----  
Out Of Memory Error  
Traceback (most recent call last)  
<ipython-input-11-af73f3eb6fa7> in <cell line: 3>()  
    1 import os  
    2 os.environ["WANDB_MODE"] = "disabled"  
--> 3 trainer.train()  
  
-----  
^ 36 frames -----  
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/linear.py in forward(self, input)  
123  
124     def forward(self, input: Tensor) -> Tensor:  
--> 125         return F.linear(input, self.weight, self.bias)  
126  
127     def extra_repr(self) -> str:  
  
Out Of Memory Error: CUDA out of memory. Tried to allocate 392.00 MiB. GPU 0 has a total capacity of 39.56 GiB of which 360.81 MiB is free. Process 7258 has 39.20 GiB memory in use. Of the allocated memory 36.62 GiB is allocated by PyTorch, and 2.09 GiB is reserved by PyTorch but unallocated. If reserved but unallocated memory is large try setting PYTORCH_CUDA_ALLOC_CONF=expandable_segments=True to avoid fragmentation. See documentation for Memory Management (https://pytorch.org/docs/stable/notes/cuda.html#environment-variables)
```

# LLama Prompt Optimizations

Used various different prompts to see the effect on the model:

```
prompts = [
    "Predict the geolocation of this image based on its visual cues."
    "Think step by step, first guessing the country, then region, then the city/region, "
    "then the latitude and longitude. Please return an exact latitude and longitude in this format: (latitude, longitude).",
    "Based on this image, predict the latitude and longitude by first identifying its broad region (e.g., North America, Europe, etc.). "
    "Then narrow down to the specific country, city, and exact location. Output only the latitude and longitude in this format: (latitude, longitude).",
    "Analyze the visual cues in this image and predict its exact geolocation. "
    "Start by identifying the country and region, then provide your confidence level for the specific location. "
    "Finally, output the latitude and longitude in this format: (latitude, longitude).",
    "Example: Eiffel Tower. Paris, France. Latitude: 48.858844, Longitude: 2.294351. "
    "Now, predict the geolocation of this image: Provide the latitude and longitude in this format: (latitude, longitude).",
    "Give me the latitude and longitude of the image"
]
```



# Easy and Medium Case

Easy Case



This is an image of the Statue of Liberty in New York, USA. Its approximate geographic coordinates are:

Latitude: 40.6892° N

Longitude: 74.0445°



Medium Case

Model Description: I'm a large language model, I don't have the ability to see or access images. However, I can help you find the location of the image if you provide me with more information about it.

VS.

\*\*Step 1: Guess the country\*\*

Based on the architecture, road signs, and general appearance of the buildings, I'm going to guess that the image is from a country in Eastern Europe.

\*\*Step 2: Guess the region\*\*

Romania has a diverse geography, but the image suggests a rural or suburban area with a mix of old and new buildings.

\*\*Step 3: Guess the city/region\*\*

Within Transylvania, I notice that the buildings seem to be a mix of old and new, with some traditional stone houses and others more modern structures.

\*\*Step 4: Guess the latitude and longitude\*\*

Based on the visual cues, I'm going to take a guess that the latitude and longitude are:

(45.7833, 23.7833)



# Hard Case



Prompt: Predict the geolocation of this image based on its visual cues. Think step by step, first guess.

Model Description: To predict the geolocation of the image, I'll analyze the visual cues step by step.

1. \*\*Country\*\*: The image shows a rural landscape with a road, trees, and a distant horizon. The vegetation appears to be typical of Central Europe.
2. \*\*Region\*\*: Poland has a diverse geography, but the image's landscape and road type suggest a region like Podlasie or Lublin.
3. \*\*City/Region\*\*: The image shows a rural area, but the presence of a road and trees suggests a region where such infrastructure exists, like the countryside around Bialystok.
4. \*\*Latitude and Longitude\*\*: Based on the visual cues, I'm going to estimate the latitude and longitude.

\* \*\*Latitude\*\*: 53.2° N (Bialystok is located at a latitude of approximately 53.2° N)  
\* \*\*Longitude\*\*: 23.2° E (Bialystok is located at a longitude of approximately 23.2° E)

Therefore, the predicted geolocation of the image is:

(53.2° N, 23.2° E)

VS.

Model Description: I don't have the ability to access the location of the image.

---

## Llama Run Times

One Prompt:  
7 hours for 15k images

All the prompts:  
Approx. 22 hours for 15k images



# Best Metrics for LLama Model

Prompt:

"Predict the geolocation of this image based on its visual cues. "

"Think step by step, first guessing the country, then region, then the city/region, "

"then the latitude and longitude. Please return an exact latitude and longitude in this format: (latitude, longitude)."

## Accuracy Thresholds (15,000 samples)

- Within 25 km: 8.41%
- Within 100 km: 16.3%
- Within 500 km: 48.5%

- Haversine Distances

- Mean: 2406.80 km
- Median: 532.16 km
- Minimum: 2.44 km
- Maximum: 19295.35 km
- Std deviation: 3983.53 km

# Code Snippets

```
from transformers import MllamaForConditionalGeneration, AutoProcessor, BitsAndBytesConfig
from peft import LoraConfig, get_peft_model
import torch

ckpt = "meta-llama/Llama-3.2-11B-Vision"
USE_LORA = True
GRADIENT_CHECKPOINTING = True
QUANTIZATION_8BIT = True
FREEZE_LLM = False
FREEZE_IMAGE = False

bnb_config = BitsAndBytesConfig(
    load_in_8bit=QUANTIZATION_8BIT,
    llm_int8_threshold=6.0,
    llm_int8_has_fp16_weight=False
)

if USE_LORA:
    lora_config = LoraConfig(
        r=8,
        lora_alpha=8,
        lora_dropout=0.1,
        target_modules=['down_proj', 'o_proj', 'k_proj', 'q_proj', 'gate_proj', 'up_proj', 'v_proj'],
        use_dora=True, # optional DoRA
        init_lora_weights="gaussian"
    )

model = MllamaForConditionalGeneration.from_pretrained(
    ckpt,
    torch_dtype=torch.bfloat16,
    device_map="auto",
    quantization_config=bnb_config
)

model = get_peft_model(model, lora_config)
model.print_trainable_parameters()
```

```
for prompt in prompts:
    with torch.no_grad():
        description = model.answer_question(
            image, prompt, tokenizer, max_new_tokens=512, do_sample=True, temperature=0.3
        )

    print(f"Prompt: {prompt}")
    print(f"Model Description: {description}")

    predicted_coords = extract_lat_lon(description)
    print("Predicted coords:", predicted_coords)
    print(image_path)

    if predicted_coords is not None:
        haversine_distance = haversine(predicted_coords, target)

        stats[prompt]["total_haversine"] += haversine_distance
        Loading... pt]["count"] += 1
        stats[prompt]["distances"].append(haversine_distance)

    print(f"Prompt '{prompt}' - Processed {stats[prompt]['count']} samples, Haversine Distance: {haversine_distance:.2f} km")

else: # full ft
    model = MllamaForConditionalGeneration.from_pretrained(ckpt,
        torch_dtype=torch.bfloat16, device_map="auto")

if GRADIENT_CHECKPOINTING:
    model.gradient_checkpointing_enable()

processor = AutoProcessor.from_pretrained(ckpt)
```

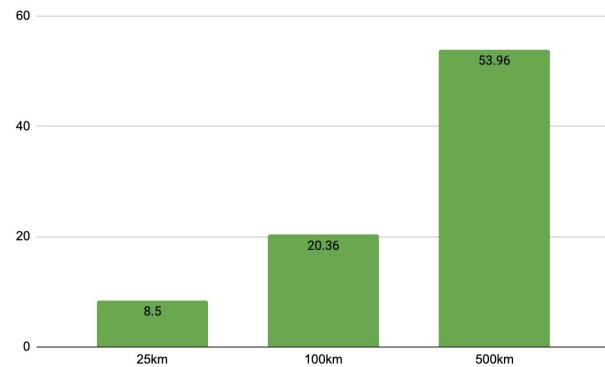
---

# Model Performance: Pixtral-12B

---

# Pixtral Base Benchmarking

- Accuracy Thresholds
  - Within 25 km: 8.5%, **increased** by 3.4%
  - Within 500 km: 20.36%, **increased** by 13.13%
  - Within 1000 km: 53.96%, **increased** by 37.79%
- Haversine Distances
  - Mean: 2719 km, **lower** by 2711
  - Median: 2587.077 km, **lower** by 1789
  - Minimum: 2 km, **higher** by 1.6
  - Maximum: 17837. km, **lower** by ~1k
  - Std deviation: 4623 km, **lower** by 300





Actual:  
Hoquiam, Washington



Predicted:  
Kolónia, Slovakia



Actual:  
Washington, DC



Actual:  
Cemetery in Washington, DC

# Code Snippets

## Predictions:

```
def run():
    hf_token = read_hf_token("hf_token.txt")
    model_name = "mistralai/Pixtral-12B-Base-2409"
    sampling_params = SamplingParams(max_tokens=100)

    llm = LLM(model=model_name, tokenizer_mode="mistral")

    prompt = """Predict the geolocation of this image based on its visual cues. Think step by step, first guessing
    Please return an exact latitude and longitude in this format: (latitude, longitude)"""

    tokenizer = llm.llm_engine.tokenizer.tokenizer.mistral.instruct_tokenizer

    ds = load_dataset("dalle-mini/YFCC100M_OpenAI_subset", trust_remote_code=True, split="train", streaming=True)
    iterator = iter(ds)

    counter = 0
    while (counter < 1000):
        data = next(iterator)
```

## Fine Tuning:

```
assistant_responses = []
for example in examples:
    img_path = example["image"]
    try:
        image = Image.open(img_path).convert("RGB")
    except Exception as e:
        print(f"Error loading image {img_path}: {e}")
        raise
    question = (
        f"Based on the visual cues in this image, identify the likely location. "
        f"First, specify the country, "
        f"then the region, "
        f"and finally provide approximate latitude and longitude."
    )
    answer = example["caption"]
    messages = [
        {
            "role": "user",
            "content": [
                {
                    "type": "text",
                    "text": question,
                },
                {
                    "type": "image",
                },
            ],
        },
        {
            "role": "assistant",
            "content": [

```

---

# Model Performance: Qwen2

---

# Qwen2-VL-2B-Instruct Training

**Training Results:** 5 hours training time total with 6 epochs, benchmarking 30 minutes

Predictions:

- within 25km: 7.6%, **increased** by 3.7
- within 500km: 45.5%, **increased** 33.7
- within 1000km: 62.1%, **increased** by 48.4

Distance Metrics (km):

- Mean: 2308.44, **lower** by 5305.96
- Median: 584.07, **lower** by 8152.5
- Std Dev: 4120.09, **lower** by 179.42
- Min: 5.46, **increased** by 4.92
- Max: 18296.21, **lower** by 18296.21

Best scored image (before fine tuning):



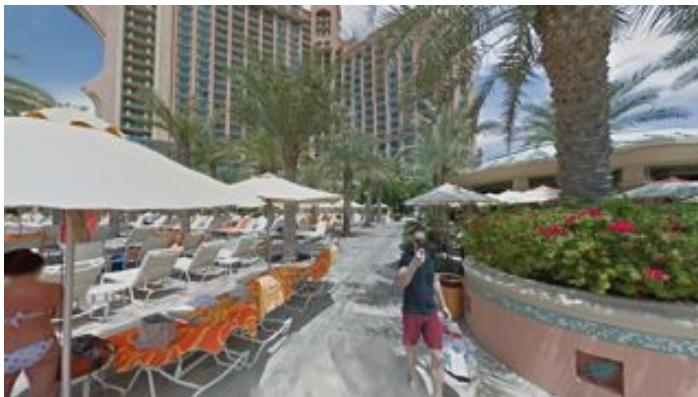
Best scored image (after):



---

# Easy and Medium Case

Easy



UAE

"latitude: 25.338888881835938, longitude: 55.338888818359375, country\_name: United Arab Emirates"

Medium Case

Spain



"The image depicts a highway in the Netherlands. Based on the visual cues, I would predict the following:

1. Country: Netherlands
2. Region: Netherlands
3. City/Region: Netherlands
4. Latitude: 52.08882885131836, longitude: 4.822288288824463, country\_name: Netherlands"

---

## Hard cases



*France*

"The image depicts a tunnel with graffiti on the walls. Based on the visual cues, it appears to be a tunnel in a city in the United States. Here is the predicted geolocation:

latitude: 33.888888818359375, longitude: -117.58888818359375, country: United States"

# Code Snippets

Predictions:

```
    content
try:
    message = [{"role": "user", "content": [
        {
            "type": "image",
            "image": og_image
        },
        {"type": "text", "text": prompt}
    ]}]
    text = processor.apply_chat_template(message, tokenize=False, add_generation_prompt=True)
    image_inputs, video_inputs = process_vision_info(message)
except (PIL.UnidentifiedImageError, Exception) as e:
    print(f"Skipped: {e}")
    continue
inputs = processor(text=[text], images=image_inputs, padding=True, return_tensors="pt").to(device)
#generate output
with torch.no_grad():
    generated_ids = model.generate(**inputs, max_new_tokens=128)
    generated_ids_trimmed = [
        out_ids[len(in_ids) :] for in_ids, out_ids in zip(inputs.input_ids, generated_ids)
    ]
    pred_text = processor.batch_decode(
        generated_ids_trimmed, skip_special_tokens=True, clean_up_tokenization_spaces=False
    )
```

Fine Tuning:

```
caption_coords = f"latitude: {data['latitude']}, longitude: {data['longitude']}, country_name: {data['country_name']}"
message = [{"role": "user", "content": [
    {
        "type": "image",
        "image": og_image
    },
    {"type": "text", "text": caption_coords}
]}]
text = processor.apply_chat_template(message, tokenize=False, add_generation_prompt=True)
image_inputs, _ = process_vision_info(message)

inputs = processor(
    text=text,
    images=image_inputs,
    padding=True,
    return_tensors="pt"
).to(device)

labels = inputs.input_ids.clone()
labels[labels == processor.tokenizer.pad_token_id] = -100

outputs = model(**inputs, labels=labels)
loss = outputs.loss
loss.backward()

optimizer.step()
lr_scheduler.step()
optimizer.zero_grad()
return loss.item()
```

---

# Case Comparisons Between Models

---

# Easy Case Pictures by Model

Phi 3.5:



Qwen:



Llama:



Pixtral:



---

## Comparative Results: Easy Cases

Image + Performance	Qwen Easy	Phi 3.5 Easy	Llama Easy	Pixtral Easy
Qwen	5 . 46 KM	5 . 66 KM	11.58 KM	111.20 KM
Phi 3.5	5985.50 KM	3 . 29 KM	16097.31 KM	14130.39 KM
Llama	705.62 KM	8 . 87 KM	12.6 KM	301.71 KM
Pixtral	482.7 KM	847.7 KM	2657.3 KM	298.2 KM

---

# Medium Case Pictures by Model

Phi 3.5:



Llama:



Qwen:



Peru (location\_081482)

Pixtral:



---

## Comparative Results: Medium Cases

Image + Performance	Qwen Medium	Phi 3.5 Medium	Llama Medium	Pixtral Medium
Qwen	1109.59 KM	43.11 KM	20.70 KM	421.69 KM
Phi 3.5	1433.74 KM	86.34 KM	32.95 KM	1240.71 KM
Llama	1327.68 KM	103.21 KM	190.4 KM	482.11 KM
Pixtral	1131.2 KM	658.5 KM	765 KM	590.2 KM

---

## Hard Case Pictures by Model

Phi 3.5:



Qwen:



Llama:



Pixtral:



---

## Comparative Results: Hard Cases

Image + Performance	Qwen Hard	Phi 3.5 Hard	Llama Hard	Pixtral Hard
Qwen	15736.92 KM	1046.49 KM	50.74 KM	1123.49 KM
Phi 3.5	4463.93 KM	16333.86 KM	459.46 KM	6396.82 KM
Llama	7809.10 KM	3627.88 KM	14483.92 KM	1337.09 KM
Pixtral	14026.1 KM	3957.1 KM	545.5 KM	18816.1 KM

---

## Comparative Performance Takeaways: Phi 3.5

One of the main observation here was that it performed relatively poorly on the others easy cases, but it performed better than others (Llama and Pixtral) on medium and hard cases. The model is able to pick up smaller details which could lead it to understand overall locations and regions better. However, since it is a smaller LLM, it may not have the background information that allows the other models to narrow down its location in easy cases. This can include small information like vegetation or climate by location, area codes, or more. To improve this model, one key aspect would be to provide it necessary background information in the prompts, or introduce this further background knowledge directly in the training. Overall the model is able to perform relatively well compared to its counterparts, but it is not the best performing model. There are a few edge cases where it outperforms the other models, but the general trend is that it is less accurate.

---

## Comparative Performance Takeaways: Llama

One thing with Llama was that it was a very large LLM which meant that it could handle a lot of obscure information which it could relate to different instances. This proved to be both a positive and a negative. On one hand the model could pick up different small factors such as billboards, road direction, and road signs but on the other hand if multiple cities had similar features the model would mix them up leading to differing results. Sometimes these results were very far from the true distance. The Llama performed exceptionally well in the easy and medium cases but sometimes struggled to understand the hard cases. This was due to the model randomly guessing the location. One other thing noticed was the importance of the prompt given to the model. After testing on different models, all looking to get the longitude and latitude, they were performing significantly different based on the prompt that was given. After providing a direction to the model for the necessary steps starting from a broad range such as country to narrowing it down slowly to coordinates, the model could understand the desired output much better than the others.

---

## Comparative Performance Takeaways: Pixtral

Pixtral 12B, overall, performed okay. Comparing the results, Pixtral consistently performed consistently around the 1000km mark. There were very few images that it was able to guess extremely close, but it was able to usually consistently guess the region within 1000 km. While this still could be improved, the consistency factor seems to be higher than other models, like Qwen. While Qwen can be very accurate sometimes, it can also miss very wide as well, giving it low precision. On the other hand, Pixtral seems to have medium precision and medium accuracy.

Additionally, performance on areas still outside of the US and Europe after fine-tuning still seems to perform worse. It is usually able to identify European and United States areas. For example, the Australia and Singaporean coverage in the hard cases (Qwen and Pixtral, respectively) performed much worse than LLaMa's US hard case.

---

## Comparative Performance Takeaways: Qwen

Qwen2 performed relatively well on all the easy, medium, and hard cases in comparison to the other models' results. Because it has been trained on instruction and performs well on arbitrary image resolutions, it was able to handle the task of predicting geotags better than the other models. Qwen2 performed the worst on PHI-3.5's hard case potentially due because of the lack of signs or visual cues. In terms of location, Qwen2 seemed to have consistent results no matter what country or continent it was predicting. The hard cases for Qwen2 were inconsistent as Pixtral's hard case contained more imagery and visual cues while LLamas' contained less. Yet, Qwen2 was successful with the LLama image, but not the Pixtral. This may be because the Pixtral image contained more complex or ambiguous cues, making it harder for Qwen2 to decide on useful geographical patterns.

To further improve the performance, various prompts should be tested to determine what's the best way to advise the model to look at smaller details.

---

# Reflection

---

# Difficulties

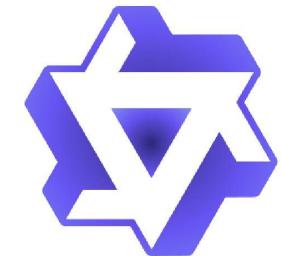
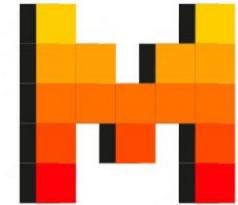
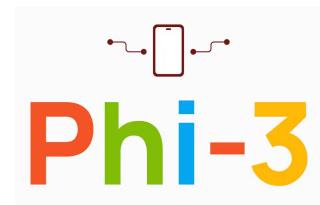
- Memory constraints, even with 40gb of GPU RAM
- Model was 24.2 GB
  - Had to be quantized to 8.7gb by lowering number precision from bf16 to float8
- Random Colab Freezing/Slowing down
- Inefficient use of GPU
- PACE Timeouts



---

# Conclusion

- We iterated to a good dataset that gives us realistic real world data with coordinates
- We test a variety of models
  - Big models like LLaMa and Pixtral
  - Smaller models like Qwen2 and Phi 3.5
  - Qwen2 already instruction-tuned
- There is improvement
  - Slight, but significant
- Methodologies can be iterated upon
- Ensemble strategies to be implemented
  - Bagging
  - Boosting



---

## Lessons Learnt + Good Points:

- Prompting can make a big difference: Small changes in prompts impacted the model's ability to return a location or a more accurate location
- Dataset Quality is Important: The initial dataset did not have much street view images, and after pivoting to a new dataset, we saw better performance.
- More Data is Needed: Although this is a start, to get more precise location predictions, we need a lot more data through images and background information.
- Fine Tuning Can be Resource Intensive: We saw several issues early on with the size of the models being a limitation with the systems we were using.
- All Models Improved Performance: After fine tuning, each model had improved upon its baseline benchmarking metrics.

---

## Future Work:

- Parallelizing Operations: By using parallel operations, we would be able to work with larger models which would allow for better performance. This would also allow us to train on larger datasets which can help improve performance.
- Incorporate Background Information: One improvement would be to incorporate background information in the prompts past just the image and question prompt. The background information could help guide the LLM
- Further Optimize the Prompt: The prompting has shown already how important small changes can be. In the future, testing more prompts could help improve performance.
- Larger and Richer Dataset: We need a much larger dataset (millions of images) to get more accurate location predictions.



# Read Me File Outline

1. Source Code:
  - a. The zip contains 4 folders, one for each model
  - b. Each folder contains code on benchmarking, fine tuning, and testing the models
  - c. Folders also contain the adapter output of the fine tuned models
2. How to Run it:
  - a. With this being a research analysis project, there is no application that results explicitly from the project.
  - b. The result of this project is the fine tuned LLMs and the comparative analysis. The fine tuned LLMs can be run using the adapter fine tuned model files. Additionally, the source code contains cells used to run the adapter fine tuned model which can be used with any picture passed in.
  - c. The ReadMe contains sub-instructions for each folder on what each file does and how it can be used to run it.