1 Appendix A: Using ColPali

Edited version of Tony Wu's notebook

1.1 Run inference with ColPali

[1]:
```python
import pprint
from io import BytesIO
from typing import List, Tuple

import matplotlib.pyplot as plt
import pandas as pd
import requests
import torch
from colpali_engine.interpretability import (
    get_similarity_maps_from_embeddings,
    plot_all_similarity_maps,
    plot_similarity_map,
)
from huggingface_hub import login
from IPython.display import display
from PIL import Image
from transformers import BatchFeature, ColPaliForRetrieval, ColPaliProcessor,␣
  ↪ProcessorMixin
```

Because ColPali uses the PaliGemma (Gemma-licensed) as its VLM backbone, you will have to login to a HuggingFace account that has accepted the terms and conditions of `google/paligemma-3b-mix-448`.

[2]:
```python
secret = pd.read_csv("secret.config", header=None)
HF_TOKEN = secret[1][1]
del secret
login(HF_TOKEN)
```

1.1.1 custom utils for images

[3]:
```python
def load_image_from_url(url: str) -> Image.Image:
    """
    Load a PIL image from a valid URL.
    """
    response = requests.get(url)
    return Image.open(BytesIO(response.content))


def scale_image(image: Image.Image, new_height: int = 1024) -> Image.Image:
    """
    Scale an image to a new height while maintaining the aspect ratio.
    """
    # Calculate the scaling factor
    width, height = image.size
    aspect_ratio = width / height
    new_width = int(new_height * aspect_ratio)

    # Resize the image
    scaled_image = image.resize((new_width, new_height))

    return scaled_image
```

1.1.2 Load the ColPali model and processor

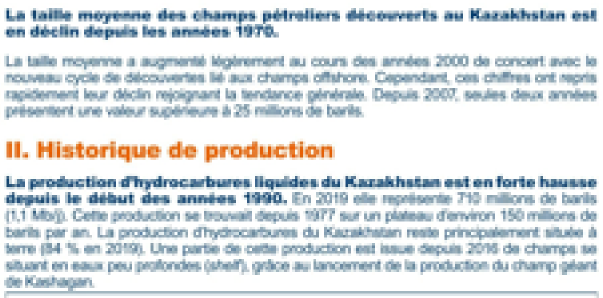
[4]:
```
model_name = "vidore/colpali-v1.2-hf"
device = "cuda:0"

model = ColPaliForRetrieval.from_pretrained(
    model_name,
    torch_dtype=torch.bfloat16,
    device_map=device,
).eval()

processor = ColPaliProcessor.from_pretrained(model_name)
```

```
Loading checkpoint shards:   0%|          | 0/2 [00:00<?, ?it/s]
```

1.1.3 Image and query inputs

[5]:
```
images: List[Image.Image] = [
    load_image_from_url(
        "https://github.com/tonywu71/colpali-cookbooks/blob/main/examples/data/
 ↪shift_kazakhstan.jpg?raw=true"
    ),
    load_image_from_url(
        "https://github.com/tonywu71/colpali-cookbooks/blob/main/examples/data/
 ↪energy_electricity_generation.jpg?raw=true"
    ),
]

queries: List[str] = [
    "Quelle partie de la production pétrolière du Kazakhstan provient de champs␣
 ↪en mer ?",
    "Which hour of the day had the highest overall electricity generation in␣
 ↪2019?",
]

# Preview the input images
for image in images:
```

```python
display(scale_image(image, 256))
```

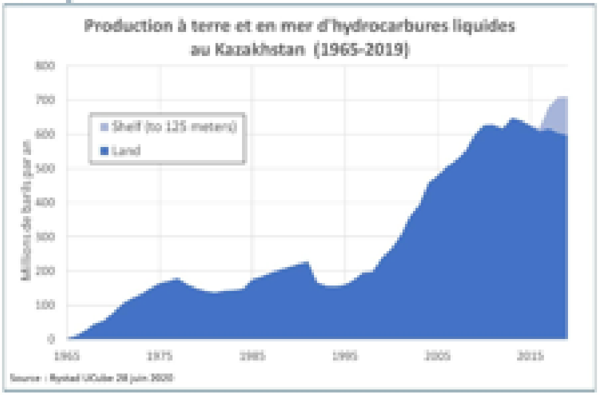


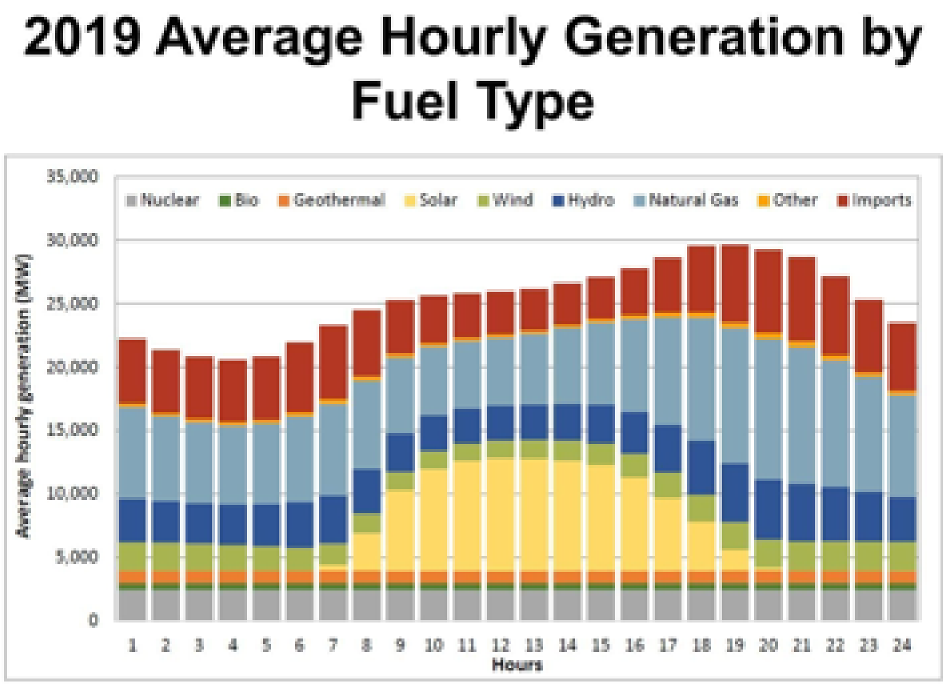
La taille moyenne des champs pétroliers découverts au Kazakhstan est en déclin depuis les années 1970.

La taille moyenne a augmenté légèrement au cours des années 2000 de concert avec le nouveau cycle de découvertes lié aux champs offshore. Cependant, ces chiffres ont repris rapidement leur déclin rejoignant la tendance générale. Depuis 2007, seules deux années présentent une valeur supérieure à 25 millions de barils.

II. Historique de production

La production d'hydrocarbures liquides du Kazakhstan est en forte hausse depuis le début des années 1990. En 2019 elle représente 710 millions de barils (1,1 Mb/j). Cette production se trouvait depuis 1977 sur un plateau d'environ 150 millions de barils par an. La production d'hydrocarbures du Kazakhstan reste principalement située à terre (84 % en 2019). Une partie de cette production est issue depuis 2016 de champs se situant en eaux peu profondes (shelf), grâce au lancement de la production du champ géant de Kashagan.





1.1.4 Preprocess and embed the queries and the images

```
[6]: # Preprocess inputs
     batch_images = processor(images=images).to(model.device)
     batch_queries = processor(text=queries).to(model.device)

     # Forward pass
     with torch.no_grad():
         image_embeddings = model(**batch_images).embeddings
         query_embeddings = model(**batch_queries).embeddings
```

1.1.5 Score the queries against the images

With the notebook's default images and queries, you should obtain a score matrix where the maximum scores are on the diagonal. This means the model has retrieved the correct image for each query.

```
[7]: scores = processor.score_retrieval(query_embeddings, image_embeddings)  #␣
      ↪(n_queries, n_images)

     scores
```

```
[7]: tensor([[20.7500, 10.3125],
             [11.7500, 16.7500]], dtype=torch.bfloat16)
```

1.2 Generate similarity maps

Here we see how to generate similarity maps between the query and the image.

```
[8]: def get_n_patches(processor: ProcessorMixin, patch_size: int) -> Tuple[int,␣
      ↪int]:
         n_patches_x = processor.image_processor.size["width"] // patch_size
         n_patches_y = processor.image_processor.size["height"] // patch_size
         return n_patches_x, n_patches_y


     def get_image_mask(processor: ProcessorMixin, batch_images: BatchFeature) ->␣
      ↪torch.Tensor:
         return batch_images.input_ids == processor.image_token_id
```

1.2.1 Inputs

To simplify the process, we will use the first query and image from the previous section.

```
[9]: image = images[0]
     query = queries[0]
     processed_query = processor(text=[query]).to(model.device)

     image_embedding = image_embeddings[[0], ...]
```

```
query_embedding = query_embeddings[[0], ...]
```

1.2.2 Get the per-token similarity maps

[10]:
```
# Get the number of patches and the image mask
n_patches = get_n_patches(processor=processor, patch_size=model.vlm.
 ↪vision_tower.config.patch_size)
image_mask = get_image_mask(processor=processor, batch_images=batch_images)

# Get the similarity maps for the first image in the batch
batched_similarity_maps = get_similarity_maps_from_embeddings(
    image_embeddings=image_embedding,
    query_embeddings=query_embedding,
    n_patches=n_patches,
    image_mask=image_mask,
)
similarity_maps = batched_similarity_maps[0]  # (query_length, n_patches_x,␣
 ↪n_patches_y)

# Use this cell output to choose a token using its index
query_content = processor.decode(processed_query.input_ids[0]).
 ↪replace(processor.tokenizer.pad_token, "")
query_content = query_content.replace(processor.query_augmentation_token, "").
 ↪strip()
query_tokens = processor.tokenizer.tokenize(query_content)

pprint.pprint({idx: val for idx, val in enumerate(query_tokens)})
```

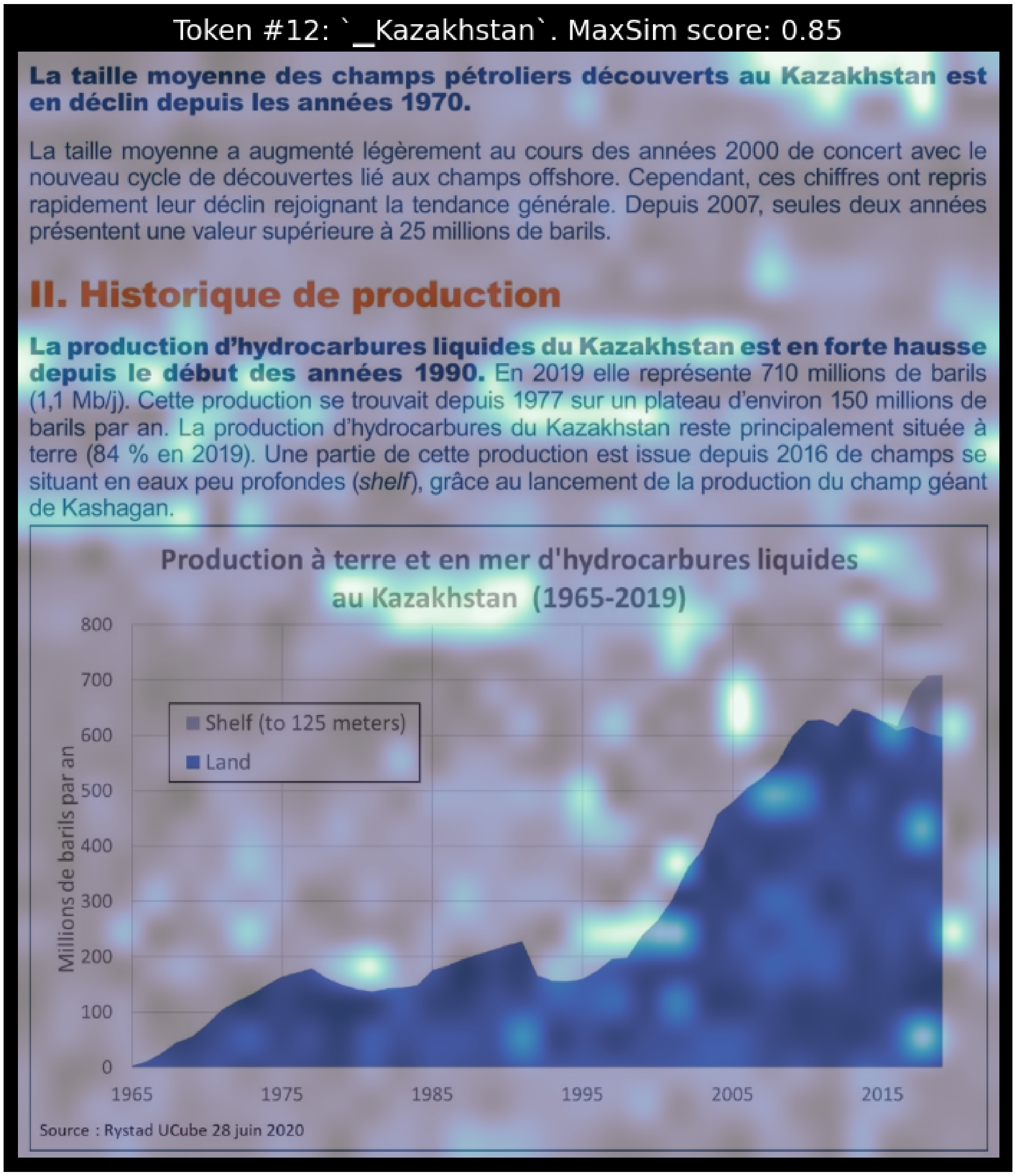
```
{0: '<bos>',
 1: 'Question',
 2: ':',
 3: ' Quelle',
 4: ' partie',
 5: ' de',
 6: ' la',
 7: ' production',
 8: ' p',
 9: 'étro',
 10: 'lière',
 11: ' du',
 12: ' Kazakhstan',
 13: ' provi',
 14: 'ent',
 15: ' de',
 16: ' champs',
 17: ' en',
 18: ' mer',
19: ' ?'}
```

1.2.3 Select the query token of interest and visualize the associated similarity map

```
[11]: # Choose a token using its index
token_idx = 12  # e.g. if "12: ' Kazakhstan',", set 12 to choose the token␣
 ↪' Kazakhstan'

print(f"Selected token: `{query_tokens[token_idx]}`")

# Retrieve the similarity map for the chosen token
current_similarity_map = similarity_maps[token_idx]  # (n_patches_x,␣
 ↪n_patches_y)

fig, ax = plot_similarity_map(
    image=image,
    similarity_map=current_similarity_map,
    figsize=(8, 8),
    show_colorbar=False,
)

max_sim_score = similarity_maps[token_idx, :, :].max().item()
ax.set_title(f"Token #{token_idx}: `{query_tokens[token_idx]}`. MaxSim score:␣
 ↪{max_sim_score:.2f}", fontsize=14)
```

```
Selected token: ` Kazakhstan`
```

```
[11]: Text(0.5, 1.0, 'Token #12: ` Kazakhstan`. MaxSim score: 0.85')
```


Token #12: `_Kazakhstan`. MaxSim score: 0.85

La taille moyenne des champs pétroliers découverts au Kazakhstan est en déclin depuis les années 1970.

La taille moyenne a augmenté légèrement au cours des années 2000 de concert avec le nouveau cycle de découvertes lié aux champs offshore. Cependant, ces chiffres ont repris rapidement leur déclin rejoignant la tendance générale. Depuis 2007, seules deux années présentent une valeur supérieure à 25 millions de barils.

II. Historique de production

La production d’hydrocarbures liquides du Kazakhstan est en forte hausse depuis le début des années 1990. En 2019 elle représente 710 millions de barils (1,1 Mb/j). Cette production se trouvait depuis 1977 sur un plateau d’environ 150 millions de barils par an. La production d’hydrocarbures du Kazakhstan reste principalement située à terre (84 % en 2019). Une partie de cette production est issue depuis 2016 de champs se situant en eaux peu profondes (*shelf*), grâce au lancement de la production du champ géant de Kashagan.

Production à terre et en mer d'hydrocarbures liquides au Kazakhstan (1965-2019)
Millions de barils par an
0
100
200
300
400
500
600
700
800
1965
1975
1985
1995
2005
2015
Shelf (to 125 meters)
Land
Source : Rystad UCube 28 juin 2020

The brighter the patch, the higher similarity score it has with the selected token. This similarity map exhibits OCR capabilities and chart understanding of ColPali. For instance, the patch containing the selected query token should be clearly highlighted in the similarity map.

1.2.4 Generate and save similarity maps for all query tokens

You can use the higher-level function `generate_similarity_maps` to generate similarity maps for all query tokens.

```
[12]: plots = plot_all_similarity_maps(
          image=image,
          query_tokens=query_tokens,
          similarity_maps=similarity_maps,
          figsize=(8, 8),
          show_colorbar=False,
          add_title=True,
      )

      for idx, (fig, ax) in enumerate(plots):
          savepath = f"similarity_map_{idx}.png"
          fig.savefig(savepath, bbox_inches="tight")
          print(f"Similarity map for token `{query_tokens[idx]}` saved at␣
       ↪`{savepath}`")

      plt.close("all")
```

```
Similarity map for token `<bos>` saved at `similarity_map_0.png`
Similarity map for token `Question` saved at `similarity_map_1.png`
Similarity map for token `:` saved at `similarity_map_2.png`
Similarity map for token ` Quelle` saved at `similarity_map_3.png`
Similarity map for token ` partie` saved at `similarity_map_4.png`
Similarity map for token ` de` saved at `similarity_map_5.png`
Similarity map for token ` la` saved at `similarity_map_6.png`
Similarity map for token ` production` saved at `similarity_map_7.png`
Similarity map for token ` p` saved at `similarity_map_8.png`
Similarity map for token `étro` saved at `similarity_map_9.png`
Similarity map for token `lière` saved at `similarity_map_10.png`
Similarity map for token ` du` saved at `similarity_map_11.png`
Similarity map for token ` Kazakhstan` saved at `similarity_map_12.png`
Similarity map for token ` provi` saved at `similarity_map_13.png`
Similarity map for token `ent` saved at `similarity_map_14.png`
Similarity map for token ` de` saved at `similarity_map_15.png`
Similarity map for token ` champs` saved at `similarity_map_16.png`
Similarity map for token ` en` saved at `similarity_map_17.png`
Similarity map for token ` mer` saved at `similarity_map_18.png`
Similarity map for token ` ?` saved at `similarity_map_19.png`
```