Retrieval Augmented Generation

Pol Pastells

January 10, 2025

This paper presents an introduction to Retrieval Augmented Generation (RAG), a technology that enhances Large Language Models by combining them with information retrieval systems. We analyze the three fundamental components of RAG systems: indexing, retrieval, and generation. We investigate multi-modal retrieval capabilities, exemplified by ColPali, through practical implementation examples. Finally, we discuss RAG's position in the evolving landscape of information retrieval, particularly in relation to emerging long-context language models.

Contents

1 *Introduction*

Large Language Models (LLMs) have demonstrated remarkable capabilities in generating human-like text and reasoning about diverse topics. However, they face significant limitations: their knowledge is static (frozen at training time), they can hallucinate or generate incorrect information, and they lack transparency in their responses. Furthermore, they cannot access private or organization-specific information unless it was part of their training data.

Retrieval Augmented Generation (RAG) emerged as a powerful solution to these challenges. RAG combines the generative capabilities of LLMs with dynamic information retrieval systems, allowing models to ground their responses in specific documents or knowledge bases. This approach offers several key advantages:

- **Up-to-date information:** RAG can access and utilize the latest documents and data, unlike static LLM knowledge.
- **Verifiable responses:** By citing specific sources, RAG enables fact-checking and transparency.
- **Domain adaptation:** Organizations can leverage their private documents and knowledge bases.
- **Reduced hallucination:** Grounding responses in retrieved documents helps prevent fabricated information.

A more correct term for hallucination from the medical point of view may be **confabulation** [Smith et al., 2023].

A crucial component in RAG systems is the retrieval mechanism. While early approaches relied on traditional keyword-based search, modern systems utilize dense neural retrievers like ColBERT, that stands out for its balance between computational efficiency and retrieval quality. Colpali, building upon colBERT's architecture, provides a practical implementation that we will explore in detail through examples.

Although many types of RAG exists [Gao et al., 2023], we will focus on the fundamental components. The basic RAG pipeline consists of three main stages: indexing, retrieval, and generation, each presenting its own challenges and optimization opportunities.

2 *Indexing*

The indexing phase is fundamental to RAG systems, involving the transformation of documents into searchable representations. While seemingly straightforward, this stage involves several critical decisions and techniques that significantly impact system performance.

2.1 *Document Chunking*

Documents must be divided into meaningful segments that balance information completeness with retrieval precision. Traditional approaches often relied on simple heuristics like fixed-length chunks or paragraph breaks. However, modern systems employ more sophisticated methods, such as using language models to identify semantically coherent sections. These systems often incorporate overlapping chunks to maintain context continuity and may adapt chunk sizes based on the inherent structure of the content.

Before chunking, an input query rewriter may be used to transform the initial query to its key information. For example, it would remove greetings and punctuation.'

2.2 *Embedding Strategies*

The choice of embedding strategy affects both retrieval quality and system efficiency. Traditional dense embeddings compress document information into single vectors. While powerful and computationally efficient, the compression of hundreds of tokens into a single vector leads to information loss. This limitation becomes particularly apparent when trying to capture fine-grained details or generalization capabilities.

The Massive Text Embedding Benchmark (MTEB) maintains a comprehensive leaderboard on HuggingFace (`https://huggingface.co/spaces/mteb/leaderboard`), evaluating embedding models across 58 datasets, 8 tasks, and 112 languages. Current results suggest no single model achieves state-of-the-art performance across all tasks.

Recent advances like Contextual Document Embeddings [Morris and Rush, 2024] (CDE) challenge the traditional paradigm of encoding documents in isolation. Similar to how contextual word embeddings revolutionized NLP by considering surrounding words, CDE incorporates information from neighboring documents during embedding creation. This approach leads to richer representations that better capture document relationships and achieve superior performance, especially in out-of-domain scenarios, without requiring complex training techniques or large batch sizes.

ColBERT [Khattab and Zaharia, 2020] (Contextual Late Interaction BERT) represents another significant advancement in embedding strategy. Instead of compressing entire documents into single vectors, ColBERT maintains token-level representations. To manage the computational overhead, it employs a downcasting layer that projects each token into a low-dimensional space (typically 64 or 128 dimensions). Despite storing more vectors, ColBERT remains practical through aggressive quantization techniques, compressing vectors to just 2 bits while maintaining over 99% of performance. Some implementations further reduce storage requirements through small-scale pooling, achieving 50-75% storage reduction with minimal performance impact.

3 *Retrieval*

The retrieval phase involves finding the most relevant documents for a given query. Modern RAG systems often employ multiple retrieval methods in concert, leveraging the strengths of both classical and neural approaches.

3.1 *Classical Methods*

BM25 (Best Match 25) remains a surprisingly robust baseline in information retrieval. It is an improvement over TF-IDF. It refines the ranking process by incorporating term saturation (i.e., diminishing returns when a word appears too frequently) and document length normalization, making it more robust for information retrieval tasks than TF-IDF. Despite its simplicity, BM25 offers several advantages: it handles variable-length texts naturally, requires minimal computational resources, and avoids catastrophic failures through simple statistical analysis. The algorithm particularly shines with keyword-heavy queries and when exact matching is crucial.

SPLADE [Lassance and Clinchant, 2022] (Sparse Lexical and Expansion)[1] represents an evolution of traditional lexical matching, incorporating neural networks while maintaining the benefits of sparse representations. It learns sophisticated term weighting and expansion mechanisms, effectively inferring relevant vocabulary not present in the original query. It is often used in hybrid retrieval systems.

TF-IDF: Term Frequency-Inverse Document Frequency is a statistical measure evaluating word importance within a document relative to a corpus. It combines term frequency with inverse document frequency, which down-weights terms frequent across many documents. Despite its simplicity, it remains effective for keyword-based ranking.

PageRank: Another classical retrieval approach that complements content-based methods like BM25. While BM25 focuses on term matching, PageRank analyzes document relationships, ranking items based on their connection patterns. Originally developed for web search, its principles are now used in many retrieval systems to incorporate document authority and relevance signals beyond pure content matching.

[1] There's currently a SPLADE v2 and v3.

3.2 *Neural Retrievers*

Traditional dense retrievers compress documents into single vectors, enabling efficient similarity search through techniques like approximate nearest neighbors. While this approach has proven effective, particularly when fine-tuned for specific domains, it inherits the limitations of aggressive information compression.

ColBERT's late interaction approach (see Figure 1) represents a more sophisticated retrieval mechanism. Instead of comparing single document vectors, it computes similarities between all query and document tokens using the MaxSim operator. For each query token, it identifies the most similar document token and combines these scores to determine overall relevance. This fine-grained interaction enables more nuanced matching while maintaining practical efficiency.

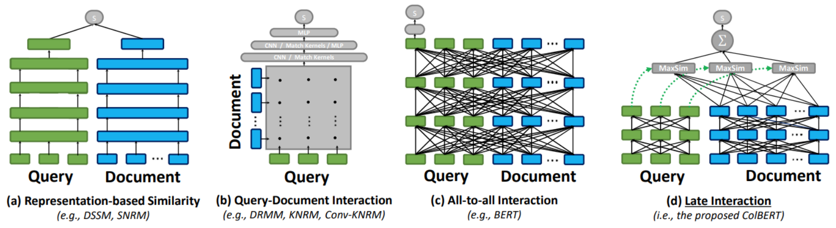

Query
Document
(a) Representation-based Similarity
(e.g., DSSM, SNRM)
MLP
CNN / Match Kernels / MLP
CNN / Match Kernels
Document
Query
(b) Query-Document Interaction
(e.g., DRMM, KNRM, Conv-KNRM)
Query
Document
(c) All-to-all Interaction
(e.g., BERT)
MaxSim
MaxSim
MaxSim
Query
Document
(d) Late Interaction
(i.e., the proposed ColBERT)

Figure 1: Query-document matching paradigms in neural information retrieval, showing the progression from bi-encoders to late interaction models. (Taken from the ColBERT paper)

3.3 *Multi-Modal Retrieval*

Recent advances, exemplified by ColPali [Faysse et al., 2024], have extended retrieval capabilities to multi-modal content, like PDFs. By applying the late-interaction approach to image tokens generated by Vision-Language Models (VLMs), these systems can seamlessly handle text, images, and tables without extensive preprocessing. As illustrated in Figure 2, traditional document retrieval pipelines involve multiple preprocessing steps: OCR for text extraction, layout detection, chunking, and embedding generation. ColPali simplifies this process by operating directly on document images through a VLM.

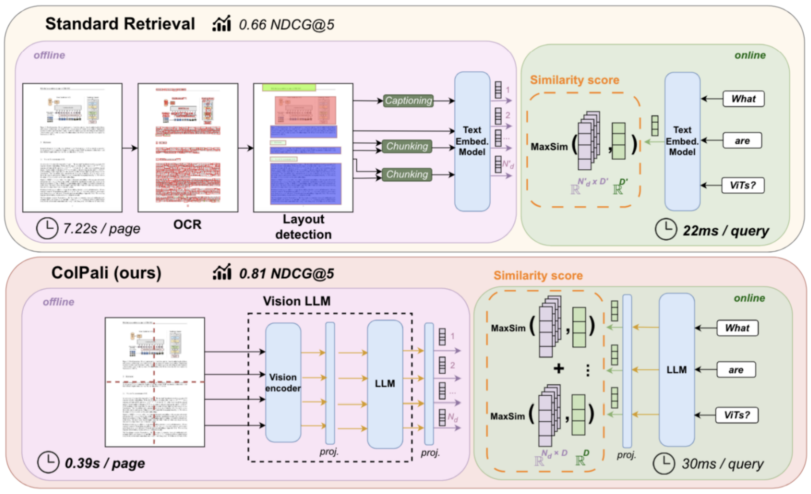


Figure 2: Comparison between traditional PDF retrieval pipeline and ColPali's simplified approach. (Taken from the ColPali paper)

3.4 *Hybrid Search*

Modern RAG systems typically combine multiple retrieval approaches to maximize effectiveness. A common pattern pairs lexical matching (like BM25) with semantic search methods. The lexical component ensures precise matching for keywords and explicit terms, while semantic search captures thematic relationships and implicit connections. Many systems add a final re-ranking step to refine results, particularly for applications requiring high precision.

GraphRAG: A specialized hybrid approach that combines knowledge graphs with LLMs. The system extracts entities, relationships, and key claims from documents to build a structured graph representation. It organizes information hierarchically, enhancing both retrieval effectiveness and result explainability.

Reranking consists on retrieving not just the top result, but the top k ones. After the retrieval a reranking step is run to obtain the result. At this step, **autocut** can be used to remove unrelated results, or to avoid comfabulations due to a low **content relevance score**, e.i., no relevant results. All of this stages can be further addapted to a specific domain.

4 *Generation*

The generation phase leverages an LLM to synthesize a coherent response using both the user prompt and the retrieved content. This step requires balancing between faithfully representing the retrieved information and crafting a relevant, contextual response that directly addresses the user's query.

4.1 *Prompt Engineering for Generation*

The structure of the generation prompt significantly impacts the quality of the response. Effective prompts typically frame the context for the LLM, provide explicit instructions for citation or attribution, and set clear guidelines for the response format and style. Additional parameters might include specific constraints such as word limits or required technical depth.

4.2 *Response Synthesis Strategies*

The LLM must employ several strategies to generate effective responses. First, it needs to integrate relevant details from multiple retrieved passages while maintaining coherence throughout the response. Second, the model must ensure source fidelity, accurately reflecting the retrieved information without distortion. Third, the generation process must maintain alignment with the original query, focusing on the specific aspects requested by the user. Finally, the model needs to maintain consistency when synthesizing information from multiple sources, avoiding contradictions or inconsistencies in the final response.

4.3 *Generation Challenges*

A primary concern is hallucination prevention, to ensure the model does not generate content unsupported by the retrieved context. Context length management presents another challenge, as the model must balance comprehensive information usage with its context window limitations. The generation process must also preserve relevance, maintaining focus on query-relevant information while filtering out tangential content. Additionally, the model must handle attribution accuracy, properly citing or referencing source materials in the generated response when required.

5 *Implementation examples*

An implementation of a RAG system described in this report is demonstrated in two Jupyter notebooks, included as appendices. Appendix A shows how to use the ColPali model with the Hugging-Face library for inference, scoring, and interpretability. Appendix B demonstrates a fine-tuning of ColPali. Both notebooks were executed in a single RTX 4090 and are available on GitHub[2].

[2] https://github.com/Pastells/RAG_colpali

6 *Discussion and Future Developments*

The landscape of information retrieval and question answering is rapidly evolving with the emergence of long-context language models (LCLMs). While RAG systems still are the industry standard for document retrieval and question answering, recent developments in LLMs with expanded context windows, such as Mistral-NeMo (128k tokens) and Llama 3 series (131k tokens), present compelling alternatives.

Research shows that LCLMs can match or exceed RAG systems' performance in many tasks, despite not being explicitly trained for retrieval[Lee et al., 2024]. However, this superior performance comes at a significant computational cost. The choice between RAG and LCLMs involves tradeoffs between accuracy, computational resources, and cost-effectiveness, as demonstrated by comparisons between the two approaches[Li et al., 2024].

RAG systems maintain several advantages that suggest their continued relevance:

- Cost-effectiveness for repeated queries on the same document set
- Ability to provide precise citations and references
- Better handling of focused, specific tasks without context overload
- Scalability to larger document collections without degradation in precision

Looking forward, hybrid approaches that combine RAG and LCLMs, such as query routing based on model self-reflection[Li et al., 2024], may offer the best of both worlds. These systems can optimize for both performance and cost, while maintaining the benefits of explicit retrieval when needed.

Rather than viewing RAG as a temporary solution, it's more accurate to consider it a complementary technology that will continue to evolve alongside advances in LLMs. The future likely lies in intelligent systems that can dynamically choose between or combine these approaches based on specific use cases and requirements.

References

Manuel Faysse, Hugues Sibille, Tony Wu, Bilel Omrani, Gautier Viaud, Céline Hudelot, and Pierre Colombo. Colpali: Efficient document retrieval with vision language models. *arXiv preprint arXiv:2407.01449*, 2024. ↑5

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023. ↑2

Omar Khattab and Matei Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 39–48, 2020. ↑3

Carlos Lassance and Stéphane Clinchant. An efficiency study for splade models. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2220–2226, 2022. ↑4

Jinhyuk Lee, Anthony Chen, Zhuyun Dai, Dheeru Dua, Devendra Singh Sachan, Michael Boratko, Yi Luan, Sébastien MR Arnold, Vincent Perot, Siddharth Dalmia, et al. Can long-context language models subsume retrieval, rag, sql, and more? *arXiv preprint arXiv:2406.13121*, 2024. ↑7

Zhuowan Li, Cheng Li, Mingyang Zhang, Qiaozhu Mei, and Michael Bendersky. Retrieval augmented generation or long-context llms? a comprehensive study and hybrid approach. *arXiv preprint arXiv:2407.16833*, 2024. ↑7

John X Morris and Alexander M Rush. Contextual document embeddings. *arXiv preprint arXiv:2410.02525*, 2024. ↑3

Andrew L Smith, Felix Greaves, and Trishan Panch. Hallucination or confabulation? neuroanatomy as metaphor in large language models. *PLOS Digital Health*, 2(11):e0000388, 2023. ↑2

1 Appendix A: Using ColPali

Edited version of Tony Wu's notebook

1.1 Run inference with ColPali

[1]:
```
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]:
```
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]:
```
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

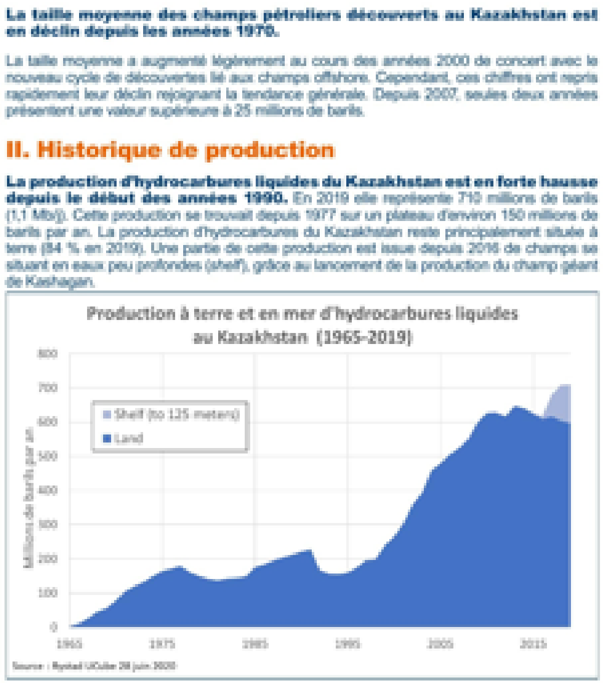
```
[4]: model_name = "vidore/colpali-v1.2-hf"
device = "cuda:0"

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:
```

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

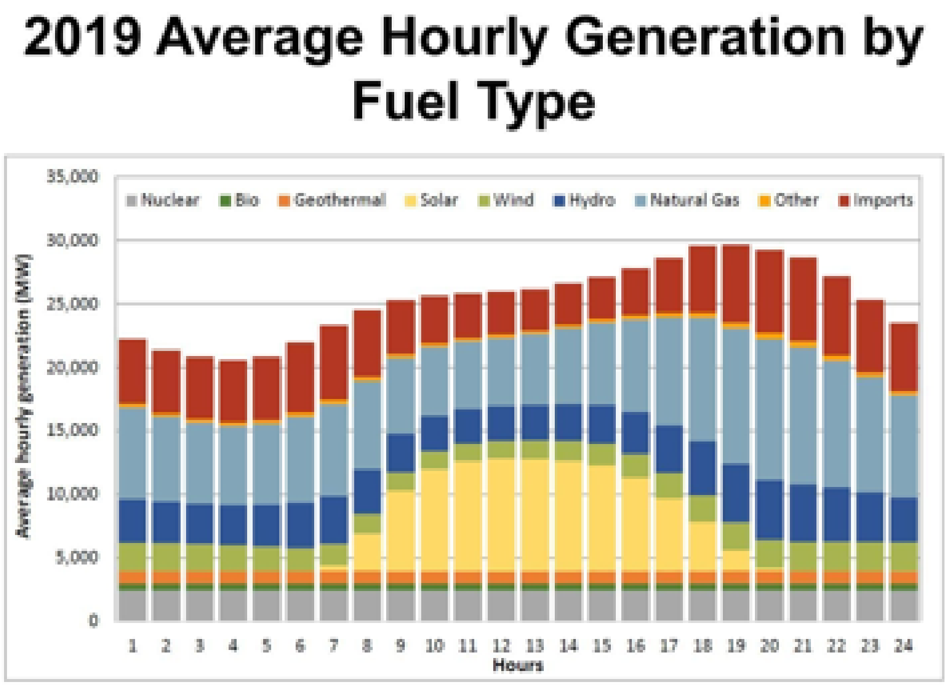


La taille moyenne des champs pétroliers découverts au Kazakhstan est en déclin depuis les années 1970.

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)})
```

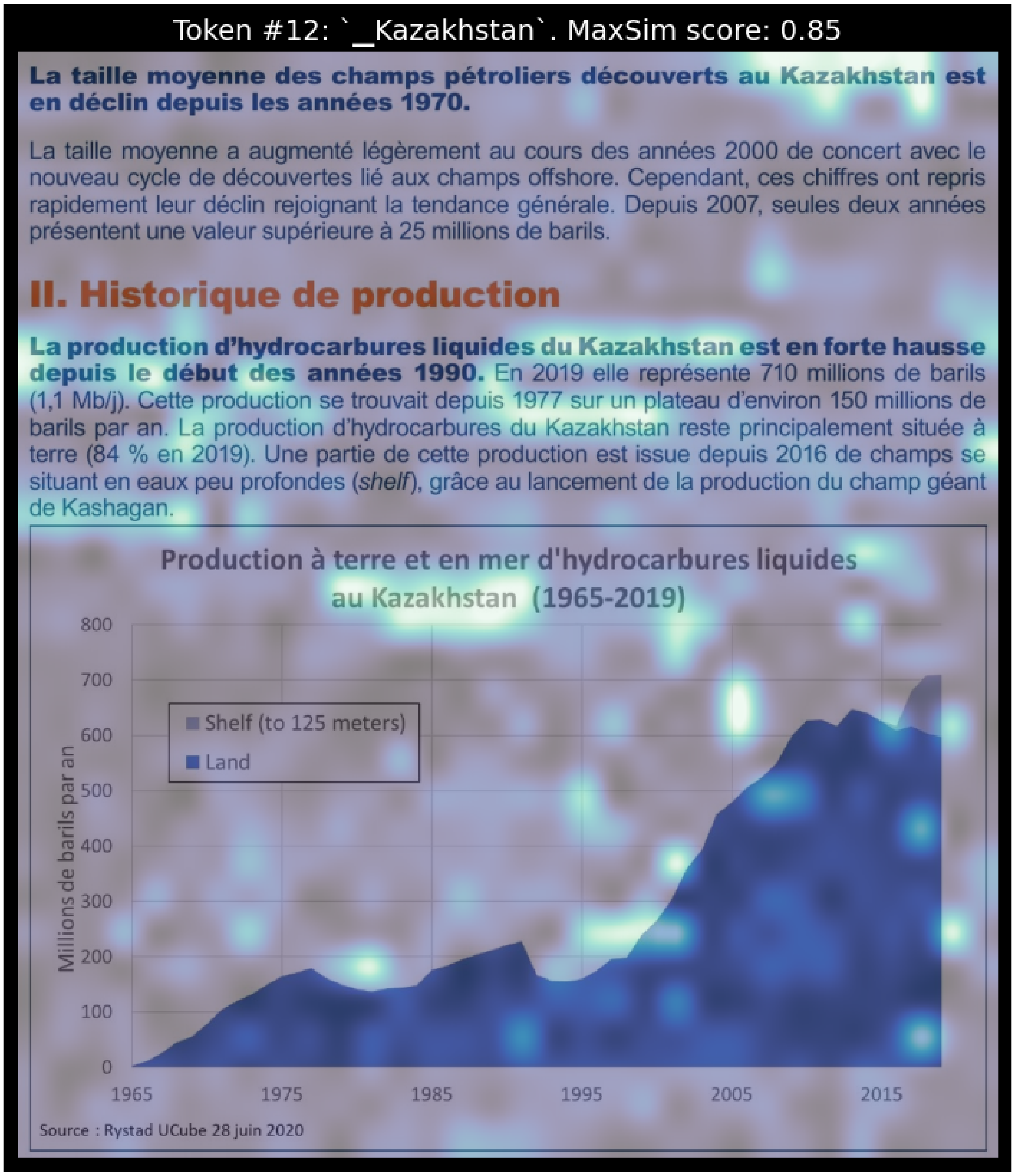
```
{0: '<bos>',
 1: 'Question',
 2: ':',
 3: ' Quelle',
 4: ' partie',
 5: ' de',
 6: ' la',
 7: ' production',
 8: ' p',
 9: 'étro',
 10: 'lière',
 11: ' du',
 12: ' Kazakhstan',
 13: ' provi',
 14: 'ent',
 15: ' de',
 16: ' champs',
 17: ' en',
 18: ' mer',
19: ' ?'}
```

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
800
700
600
500
400
300
200
100
0
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`
```

1 Appendix B: Fine-tuning ColPali

Edited version of Merve Noyan's notebook

This notebook is a very minimal example to fine-tune ColPali on UFO documents and queries a dataset synthetically generated.

Then we will show a very minimal example on how to retrieve infographics.

[1]:
```
import os

os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
os.environ["CUDA_VISIBLE_DEVICES"] = "1"
os.environ["NCCL_P2P_DISABLE"] = "1"
os.environ["NCCL_IB_DISABLE"] = "1"

import pandas as pd
import torch
from colpali_engine.loss import ColbertPairwiseCELoss
from datasets import DatasetDict, load_dataset
from huggingface_hub import login
from peft import LoraConfig, get_peft_model
from transformers import BitsAndBytesConfig, ColPaliForRetrieval,␣
 ↪ColPaliProcessor, Trainer, TrainingArguments
```

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

1.1 Loading the Model

Fine-tuning ColPali takes around 48 GB of VRAM, which is way too much for an RTX 4090. To overcome memory limits, we can apply QLoRA to only train an adapter and load the model in a lower precision (4-bit). Furthermore, we reduce the batch size and compensate for it with gradient accumulation.

[3]:
```
model_name = "vidore/colpali-v1.2-hf"

bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16,
)
model = ColPaliForRetrieval.from_pretrained(
    model_name,
    torch_dtype=torch.bfloat16,
    quantization_config=bnb_config,
    device_map="cuda:0",
).eval()

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"],
    init_lora_weights="gaussian",
)
lora_config.inference_mode = False
model = get_peft_model(model, lora_config)
processor = ColPaliProcessor.from_pretrained(model_name)
```

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

1.2 Load the dataset

We will use this dataset created by Daniel van Strien. In this blog post he explains very thoroughly how to create a dataset for retrieval tasks.

[4]:
```
ds = load_dataset("davanstrien/ufo-ColPali")
ds = ds["train"].train_test_split(test_size=0.1, seed=42)
train_ds = ds["train"]
test_ds = ds["test"]

ds
```

[4]:
```
DatasetDict({
    train: Dataset({
        features: ['image', 'raw_queries', 'broad_topical_query',
'broad_topical_explanation', 'specific_detail_query',
'specific_detail_explanation', 'visual_element_query',
'visual_element_explanation', 'parsed_into_json'],
        num_rows: 2018
    })
    test: Dataset({
        features: ['image', 'raw_queries', 'broad_topical_query',
'broad_topical_explanation', 'specific_detail_query',
'specific_detail_explanation', 'visual_element_query',
'visual_element_explanation', 'parsed_into_json'],
        num_rows: 225
    })
})
```

We need to get rid of dataset items where our text query column is None.

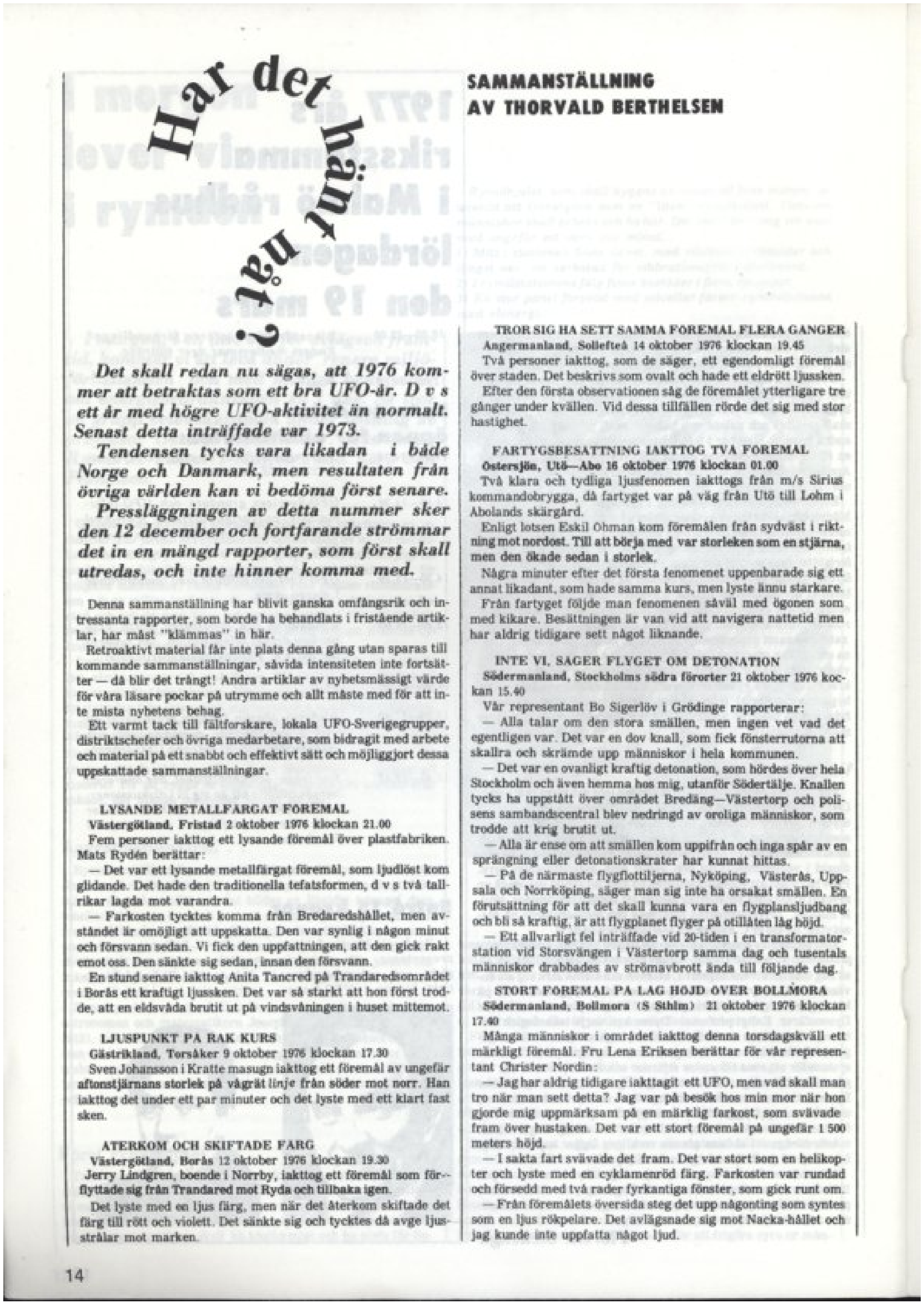
[5]:
```
train_ds = train_ds.filter(lambda example: example["specific_detail_query"] is␣
  ↪not None)
train_ds  # should be less than 2018
```

[5]:
```
Dataset({
    features: ['image', 'raw_queries', 'broad_topical_query',
'broad_topical_explanation', 'specific_detail_query',
'specific_detail_explanation', 'visual_element_query',
'visual_element_explanation', 'parsed_into_json'],
    num_rows: 1956
})
```

The dataset contains documents about UFOs and queries that might be related to document. Take alook at examples shortly.

[6]:
```
print(train_ds[0]["specific_detail_query"])
display(train_ds[0]["image"])
```

```
1976 Ufo sightings in Norway and Denmark
```



Har det hänt nåt?

SAMMANSTÄLLNING AV THORVALD BERTHELSEN

Det skall redan nu sägas, att 1976 kommer att betraktas som ett bra UFO-år. D v s ett år med högre UFO-aktivitet än normalt. Senast detta inträffade var 1973.

Tendensen tycks vara likadan i både Norge och Danmark, men resultaten från övriga världen kan vi bedöma först senare.

Pressläggningen av detta nummer sker den 12 december och fortfarande strömmar det in en mängd rapporter, som först skall utredas, och inte hinner komma med.

Denna sammanställning har blivit ganska omfångsrik och intressanta rapporter, som borde ha behandlats i fristående artiklar, har måst "klämmas" in här.

Retroaktivt material får inte plats denna gång utan sparas till kommande sammanställningar, såvida intensiteten inte fortsätter — då blir det trångt! Andra artiklar av nyhetsmässigt värde för våra läsare pockar på utrymme och allt måste med för att inte mista nyhetens behag.

Ett varmt tack till fältforskare, lokala UFO-Sverigegrupper, distriktschefer och övriga medarbetare, som bidragit med arbete och material på ett snabbt och effektivt sätt och möjliggjort dessa uppskattade sammanställningar.

LYSANDE METALLFARGAT FOREMAL

Västergötland, Fristad 2 oktober 1976 klockan 21.00

Fem personer iakttog ett lysande föremål över plastfabriken. Mats Rydén berättar:

— Det var ett lysande metallfärgat föremål, som ljudlöst kom glidande. Det hade den traditionella tefatsformen, d v s två tallrikar lagda mot varandra.

— Farkosten tycktes komma från Bredaredshållet, men avståndet är omöjligt att uppskatta. Den var synlig i någon minut och försvann sedan. Vi fick den uppfattningen, att den gick rakt emot oss. Den sänkte sig sedan, innan den försvann.

En stund senare iakttog Anita Tancred på Trandaredsområdet i Borås ett kraftigt ljussken. Det var så starkt att hon först trodde, att en eldsvåda brutit ut på vindsvåningen i huset mittemot.

LJUSPUNKT PA RAK KURS

Gästrikland, Torsåker 9 oktober 1976 klockan 17.30

Sven Johansson i Kratte masugn iakttog ett föremål av ungefär aftonstjärnans storlek på vågrät linje från söder mot norr. Han iakttog det under ett par minuter och det lyste med ett klart fast sken.

ATERKOM OCH SKIFTADE FARG

Västergötland, Borås 12 oktober 1976 klockan 19.30

Jerry Lindgren, boende i Norrby, iakttog ett föremål som förflyttade sig från Trandared mot Ryda och tillbaka igen.

Det lyste med en ljus färg, men när det återkom skiftade det färg till rött och violett. Det sänkte sig och tycktes då avge ljusstrålar mot marken.

TROR SIG HA SETT SAMMA FOREMAL FLERA GANGER

Angermanland, Sollefteå 14 oktober 1976 klockan 19.45

Två personer iakttog, som de säger, ett egendomligt föremål över staden. Det beskrivs som ovalt och hade ett eldrött ljussken.

Efter den första observationen såg de föremålet ytterligare tre gånger under kvällen. Vid dessa tillfällen rörde det sig med stor hastighet.

FARTYGSBESATTNING IAKTTOG TVA FOREMAL

Östersjön, Utö—Abo 16 oktober 1976 klockan 01.00

Två klara och tydliga ljusfenomen iakttogs från m/s Sirius kommandobrygga, då fartyget var på väg från Utö till Lohm i Abolands skärgård.

Enligt lotsen Eskil Öhman kom föremålen från sydväst i riktning mot nordost. Till att börja med var storleken som en stjärna, men den ökade sedan i storlek.

Några minuter efter det första fenomenet uppenbarade sig ett annat likadant, som hade samma kurs, men lyste ännu starkare.

Från fartyget följde man fenomenen såväl med ögonen som med kikare. Besättningen är van vid att navigera nattetid men har aldrig tidigare sett något liknande.

INTE VI, SAGER FLYGET OM DETONATION

Södermanland, Stockholms södra förorter 21 oktober 1976 klockan 15.40

Vår representant Bo Sigerlöv i Grödinge rapporterar:

— Alla talar om den stora smällen, men ingen vet vad det egentligen var. Det var en dov knall, som fick fönsterrutorna att skallra och skrämde upp människor i hela kommunen.

— Det var en ovanligt kraftig detonation, som hördes över hela Stockholm och även hemma hos mig, utanför Södertälje. Knallen tycks ha uppstått över området Bredäng—Västertorp och polisens sambandscentral blev nedringd av oroliga människor, som trodde att krig brutit ut.

— Alla är ense om att smällen kom uppifrån och inga spår av en sprängning eller detonationskrater har kunnat hittas.

— På de närmaste flygflottiljerna, Nyköping, Västerås, Uppsala och Norrköping, säger man sig inte ha orsakat smällen. En förutsättning för att det skall kunna vara en flygplansljudbang och bli så kraftig, är att flygplanet flyger på otillåten låg höjd.

— Ett allvarligt fel inträffade vid 20-tiden i en transformatorstation vid Storsvängen i Västertorp samma dag och tusentals människor drabbades av strömavbrott ända till följande dag.

STORT FÖREMAL PA LAG HÖJD ÖVER BOLLMORA

Södermanland, Bollmora (S Sthlm) 21 oktober 1976 klockan 17.40

Många människor i området iakttog denna torsdagskväll ett märkligt föremål. Fru Lena Eriksen berättar för vår representant Christer Nordin:

— Jag har aldrig tidigare iakttagit ett UFO, men vad skall man tro när man sett detta? Jag var på besök hos min mor när hon gjorde mig uppmärksam på en märklig farkost, som svävade fram över hustaken. Det var ett stort föremål på ungefär 1 500 meters höjd.

— I sakta fart svävade det fram. Det var stort som en helikopter och lyste med en cyklamenröd färg. Farkosten var rundad och försedd med två rader fyrkantiga fönster, som gick runt om.

— Från föremålets översida steg det upp någonting som syntes som en ljus rökpelare. Det avlägsnade sig mot Nacka-hållet och jag kunde inte uppfatta något ljud.

From this dataset we will have the following columns to create the documents and the queries:

- `image` contains our documents.
- `specific_detail_query` contains the textual queries.

[7]:
```python
def collate_fn(examples):
    texts = []
    images = []

    for example in examples:
        texts.append(example["specific_detail_query"])
        images.append(example["image"].convert("RGB"))

    batch_images = processor(images=images, return_tensors="pt").to(model.
↪device)
    batch_queries = processor(text=texts, return_tensors="pt").to(model.device)
    return (batch_queries, batch_images)
```

1.3 Trainer

The trainer uses a ColBERT contrastive hard-margin loss. This loss is implemented in ColPali engine, it expects batch document and query embeddings, so essentially we need to process the documents and queries separately, then pass them to the model separately, then send to loss calculation.

Note that, since we are defining a custom loss, we have to subclass transformers Trainer to be able to pass it to the model.

[8]:
```python
class ContrastiveTrainer(Trainer):
    def __init__(self, loss_func, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.loss_func = loss_func

    def compute_loss(self, model, inputs, num_items_in_batch=4,␣
↪return_outputs=False):
        query_inputs, doc_inputs = inputs
        query_outputs = model(**query_inputs)
        doc_outputs = model(**doc_inputs)
        loss = self.loss_func(query_outputs.embeddings, doc_outputs.embeddings)
        return (loss, (query_outputs, doc_outputs)) if return_outputs else loss

    def prediction_step(self, model, inputs):
        query_inputs, doc_inputs = inputs  # unpack from data collator
        with torch.no_grad():
            query_outputs = model(**query_inputs)
            doc_outputs = model(**doc_inputs)

            loss = self.loss_func(query_outputs.embeddings, doc_outputs.
↪embeddings)
```

```
            return loss, None, None
```

[9]:
```
training_args = TrainingArguments(
    output_dir="./colpali_ufo",
    num_train_epochs=1,
    per_device_train_batch_size=2,
    gradient_accumulation_steps=8,
    gradient_checkpointing=False,
    logging_steps=20,
    warmup_steps=100,
    learning_rate=5e-5,
    save_total_limit=1,
    report_to="wandb",
    dataloader_pin_memory=False,
)
```

[10]:
```
trainer = ContrastiveTrainer(
    model=model, train_dataset=train_ds, args=training_args,␣
 ↪loss_func=ColbertPairwiseCELoss(), data_collator=collate_fn
)

trainer.args.remove_unused_columns = False
```

We are training on a small dataset (little less than 2k examples) for one epoch so the training is fairly short (around 8 mins).

[11]:
```
trainer.train()
```

```
wandb: WARNING The `run_name` is currently set to the same
value as `TrainingArguments.output_dir`. If this was not intended, please
specify a different run name by setting the `TrainingArguments.run_name`
parameter.
wandb: Using wandb-core as the SDK backend.  Please refer to
https://wandb.me/wandb-core for more information.
wandb: Currently logged in as: polpastells
(clic). Use `wandb login --relogin` to force relogin

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Could not estimate the number of tokens of the input, floating-point operations
will not be computed

<IPython.core.display.HTML object>
```

```
[11]: TrainOutput(global_step=122, training_loss=0.05558847440559356,
metrics={'train_runtime': 465.0577, 'train_samples_per_second': 4.206,
'train_steps_per_second': 0.262, 'total_flos': 0.0, 'train_loss':
0.05558847440559356, 'epoch': 0.9979550102249489})
```

1.4 Load and test fine-tuned model

Let's try the fine-tuned model. You can simply test by passing in text-image pairs and check the scores for the ones that are actually pairs of each other, and also the scores of those that are irrelevant (i.e. all scores except for the scores of the matching ones).

```
[12]: print(test_ds[0]["specific_detail_query"])
display(test_ds[0]["image"])
print(test_ds[1]["specific_detail_query"])
display(test_ds[1]["image"])
print(test_ds[2]["specific_detail_query"])
display(test_ds[2]["image"])
```

```
David Copperfield special
```

NASHVILLE SCENE, Nashville, TN - Oct. 26, 1995 CR: B. Kinnard

OUT OF THIS WORLD

UFO FlyBys in Middle Tennessee

BY JOYSA M. WINTER

LaVere Pisut is well aware that some people out there think she is crazy. If they're right—if she really has hooped the loop, so to speak—she at least *knows* people think she's nuts. Her story is so bizarre, in fact, that sometimes Pisut herself can barely believe it.

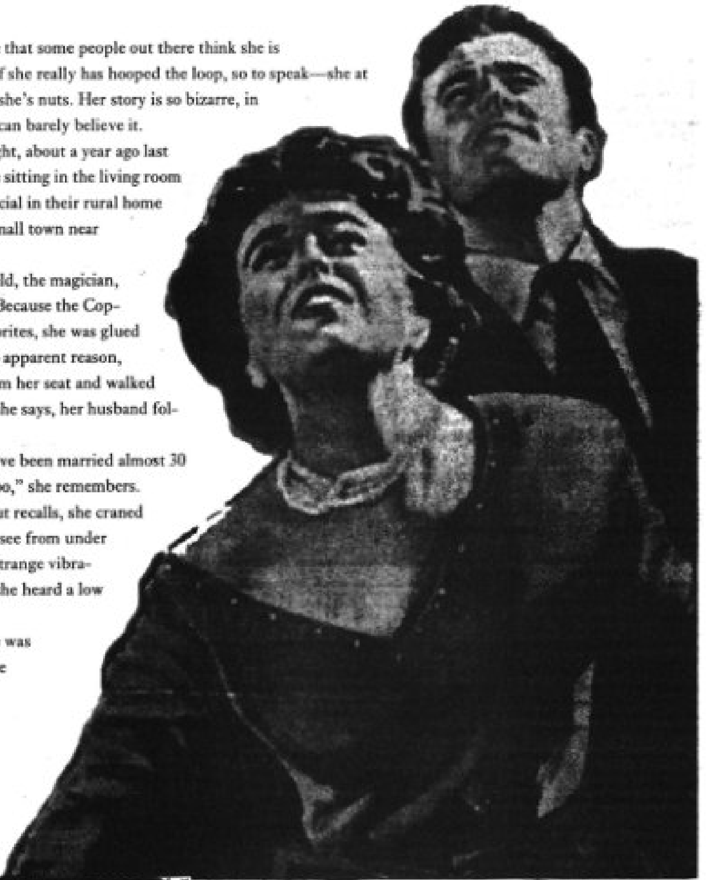
It happened one cozy spring night, about a year ago last April. Pisut and her husband were sitting in the living room watching a David Copperfield special in their rural home about 10 miles outside Baxter, a small town near Cookeville.

Pisut was excited that Copperfield, the magician, was performing some new tricks. Because the Copperfield program is one of her favorites, she was glued to the screen. Nevertheless, for no apparent reason, Pisut says, she abruptly got up from her seat and walked outside to her front porch. Soon, she says, her husband followed her outside as well.

"For him to follow me out—we've been married almost 30 years—that was kind of unusual too," she remembers.

While walking out the door, Pisut recalls, she craned her neck toward the sky, trying to see from under the eave of the porch. She felt a "strange vibration" through her shoulders, and she heard a low rumbling noise.

Her husband asked her what she was looking at, and for a moment, there was nothing to see. Until...



PHOTOS BY SUSAN ARCHER

They saw the light *Jeff Hart (left) and Marc "Pugsley" Pisut say they witnessed a strange light above the treetops near Baxter in 1992. "I swear, we weren't drunk and we weren't stoned," Pisut says.*

"Until all of a sudden, there were these lights up above the house, either square or triangular in formation," Pisut says, pointing up to a barren spot just off her porch—a rare window of sky among acres of canopied trees. "It was very dark that night, no moon, and I couldn't see stars through it. That's how I know it was solid."

Pisut doesn't know how long she and her husband sat on the porch watching the object—which was as big as her house—hover in the air. But she guesses they stood there for about 20 minutes. And now, thinking back on it, all she can do is despair over the things she wishes she had done.

"Why didn't we turn out the porch light to see it better? Or grab a flashlight and shine it up, because that's how low it was. Or we could have run off the porch to get a better look. I don't know why we didn't do these things!"

Once the object flew away across the trees, Pisut says, there was a great flash of white light and then total silence. It was a queer stillness that blanketed the countryside.

"This was spring, and spring is not quiet," she says. "We had 150 chickens and roosters, 11 geese, five dogs, and there was no sound. I don't even know where the dogs were—and they always come up when we're outside. No frogs, no peep from the goat yard next to my house. We're next to a pond but no noise from the crickets. Absolute, dead silence."

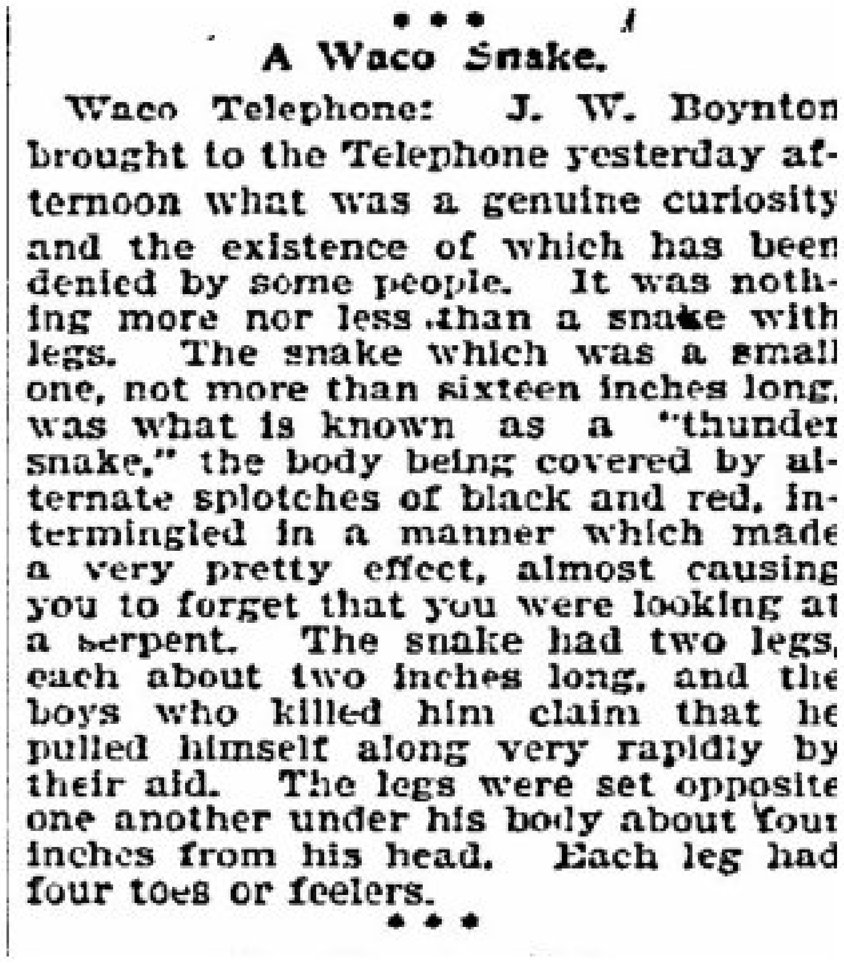
When it was all over, and Pisut had regained some of her senses, she ran inside and called her sister, who lives next door on the other side of the wooded thicket.

"LaVere called me at about 10 till 10," says Gaylene Fields, sitting on a step on her sister's front porch, staring at the blank spot in the sky where stars are now twinkling, the same spot where all this bizarreness started. "I asked her where it was headed, and my son and I got in the car and drove west toward Granville. We could see an orangey glow at the horizon, so brilliant that we thought it had crashed. Halfway there, we smelled this acrid, chemical smell we had never smelled before."

But they say they lost sight of the object. Once they had arrived in Granville, Fields and her son found some Jackson County police and a sheriff who followed them back to her sister's house. An hour later, two Putnam

(continued on page 6)

Snake with legs and feet reported in Africa, 1899



A Waco Snake.

Waco Telephone: J. W. Boynton brought to the Telephone yesterday afternoon what was a genuine curiosity and the existence of which has been denied by some people. It was nothing more nor less than a snake with legs. The snake which was a small one, not more than sixteen inches long, was what is known as a "thunder snake," the body being covered by alternate splotches of black and red, intermingled in a manner which made a very pretty effect, almost causing you to forget that you were looking at a serpent. The snake had two legs, each about two inches long, and the boys who killed him claim that he pulled himself along very rapidly by their aid. The legs were set opposite one another under his body about four inches from his head. Each leg had four toes or feelers.

Above: Fort Worth Morning Register August 12th 1899

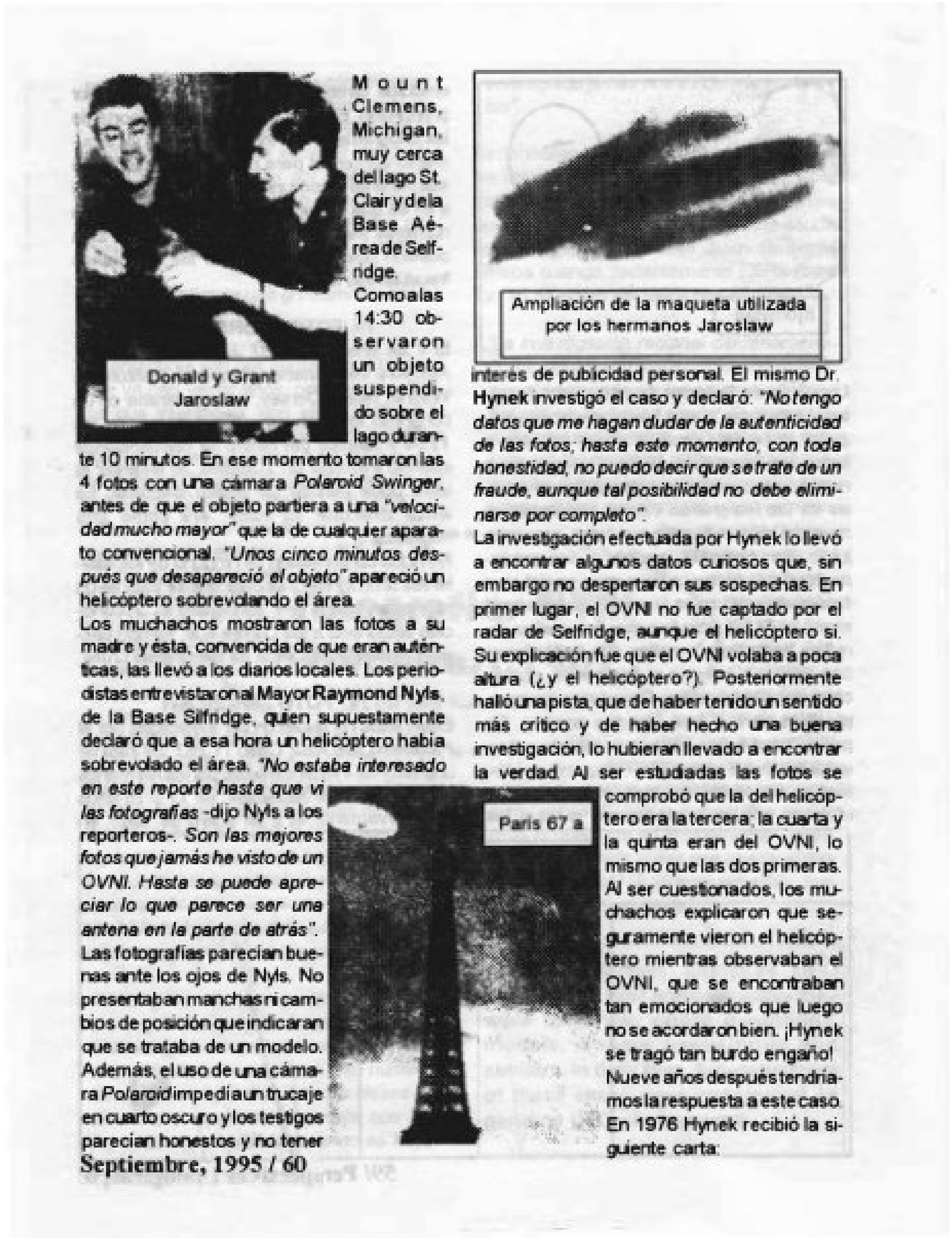
SNAKE WITH LEGS AND FEET IS REPORTED TO BE FOUND IN AFRICA

The Sacramento Bee July 20th 1950

"A snake with four legs,feet and joints is reported to have been killed by the district commissioner in Kalabo, a distant area in the North Rhodesian bush. Many strange tales of unusual snakes have come from this district in the past. Whistling snakes have been reported and once the natives insisted they had found a snake which could sing."

71

Document mentioning Dr. Hynek and the OVNI



Donald y Grant Jaroslaw

Mount Clemens, Michigan, muy cerca del lago St. Clair y de la Base Aérea de Selfridge. Como a las 14:30 observaron un objeto suspendido sobre el lago durante 10 minutos. En ese momento tomaron las 4 fotos con una cámara *Polaroid Swinger*, antes de que el objeto partiera a una *"velocidad mucho mayor"* que la de cualquier aparato convencional. *"Unos cinco minutos después que desapareció el objeto"* apareció un helicóptero sobrevolando el área.

Los muchachos mostraron las fotos a su madre y ésta, convencida de que eran auténticas, las llevó a los diarios locales. Los periodistas entrevistaron al Mayor **Raymond Nyls**, de la Base Silfridge, quien supuestamente declaró que a esa hora un helicóptero habia sobrevolado el área. *"No estaba interesado en este reporte hasta que vi las fotografías* -dijo Nyls a los reporteros-. *Son las mejores fotos que jamás he visto de un OVNI. Hasta se puede apreciar lo que parece ser una antena en la parte de atrás"*.

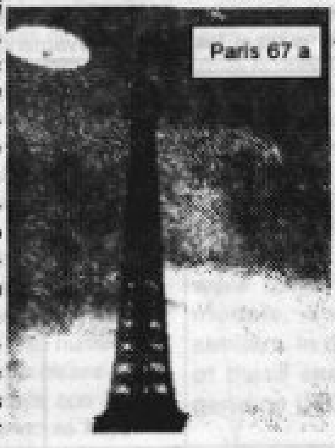
Las fotografías parecían buenas ante los ojos de Nyls. No presentaban manchas ni cambios de posición que indicaran que se trataba de un modelo. Además, el uso de una cámara *Polaroid* impedía un trucaje en cuarto oscuro y los testigos parecían honestos y no tener

Ampliación de la maqueta utilizada por los hermanos Jaroslaw

interés de publicidad personal. El mismo Dr. **Hynek** investigó el caso y declaró: *"No tengo datos que me hagan dudar de la autenticidad de las fotos; hasta este momento, con toda honestidad, no puedo decir que se trate de un fraude, aunque tal posibilidad no debe eliminarse por completo"*.

La investigación efectuada por Hynek lo llevó a encontrar algunos datos curiosos que, sin embargo no despertaron sus sospechas. En primer lugar, el OVNI no fue captado por el radar de Selfridge, aunque el helicóptero si. Su explicación fue que el OVNI volaba a poca altura (¿y el helicóptero?). Posteriormente halló una pista, que de haber tenido un sentido más crítico y de haber hecho una buena investigación, lo hubieran llevado a encontrar la verdad. Al ser estudiadas las fotos se comprobó que la del helicóptero era la tercera; la cuarta y la quinta eran del OVNI, lo mismo que las dos primeras. Al ser cuestionados, los muchachos explicaron que seguramente vieron el helicóptero mientras observaban el OVNI, que se encontraban tan emocionados que luego no se acordaron bien. ¡Hynek se tragó tan burdo engaño!

Nueve años después tendríamos la respuesta a este caso. En 1976 Hynek recibió la siguiente carta:


Paris 67 a

[13]:
```
images = [test_ds[0]["image"], test_ds[1]["image"], test_ds[2]["image"]]
texts = [test_ds[0]["specific_detail_query"],␣
 ↪test_ds[1]["specific_detail_query"], test_ds[2]["specific_detail_query"]]

# process
batch_images = processor(images=images).to(model.device)
batch_queries = processor(text=texts).to(model.device)

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

# Score the queries against the images
scores = processor.score_retrieval(query_embeddings, image_embeddings)
```

The matching text-image scores are on the diagon of the scores below, as you can see, they're matched correctly!

[14]:
```
scores
```

[14]:
```
tensor([[12.9375,  4.6250,  7.4375],
        [ 9.9375, 19.3750,  9.1875],
        [ 8.7500,  7.7188, 17.3750]], dtype=torch.bfloat16)
```