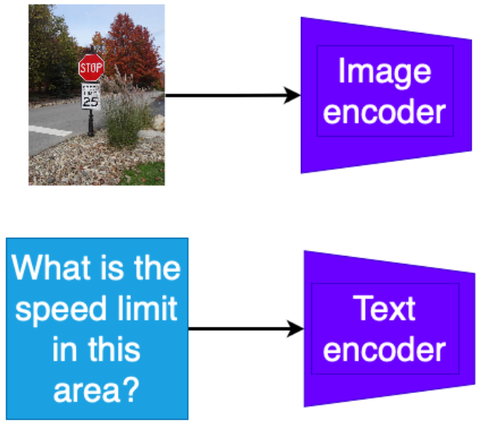
Visual question-answering (VQA)

MULTI-MODAL MODELS WITH HUGGING FACE

James Chapman
Curriculum Manager, DataCamp

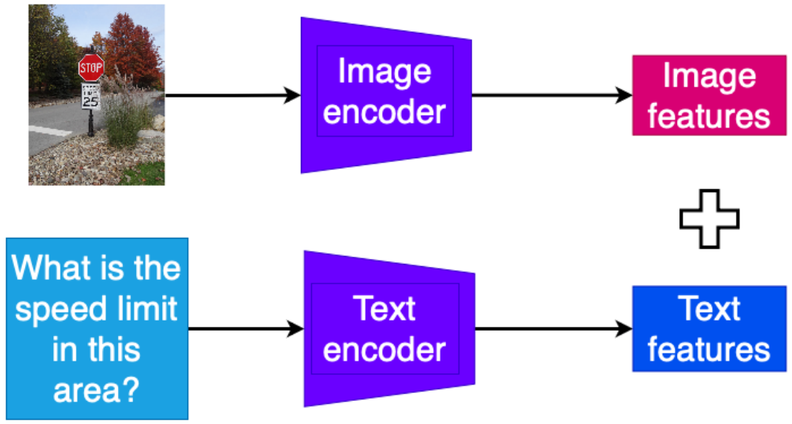
datacamp

Multimodal QA tasks



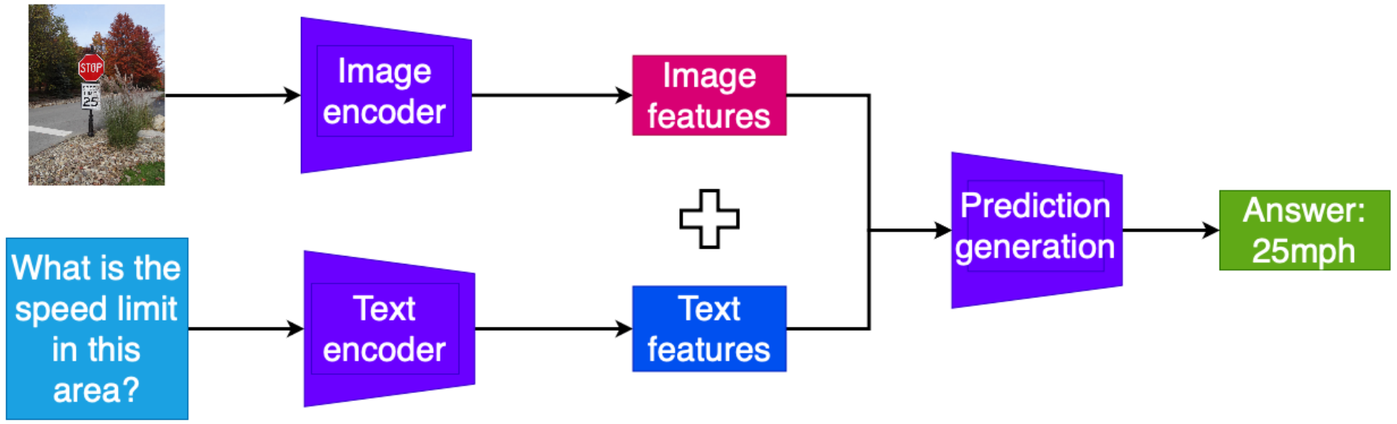
1. Separate encoding of question text and other modality

Multimodal QA tasks



1. Separate encoding of question text and other modality
2. Combination of encoded features

Multimodal QA tasks



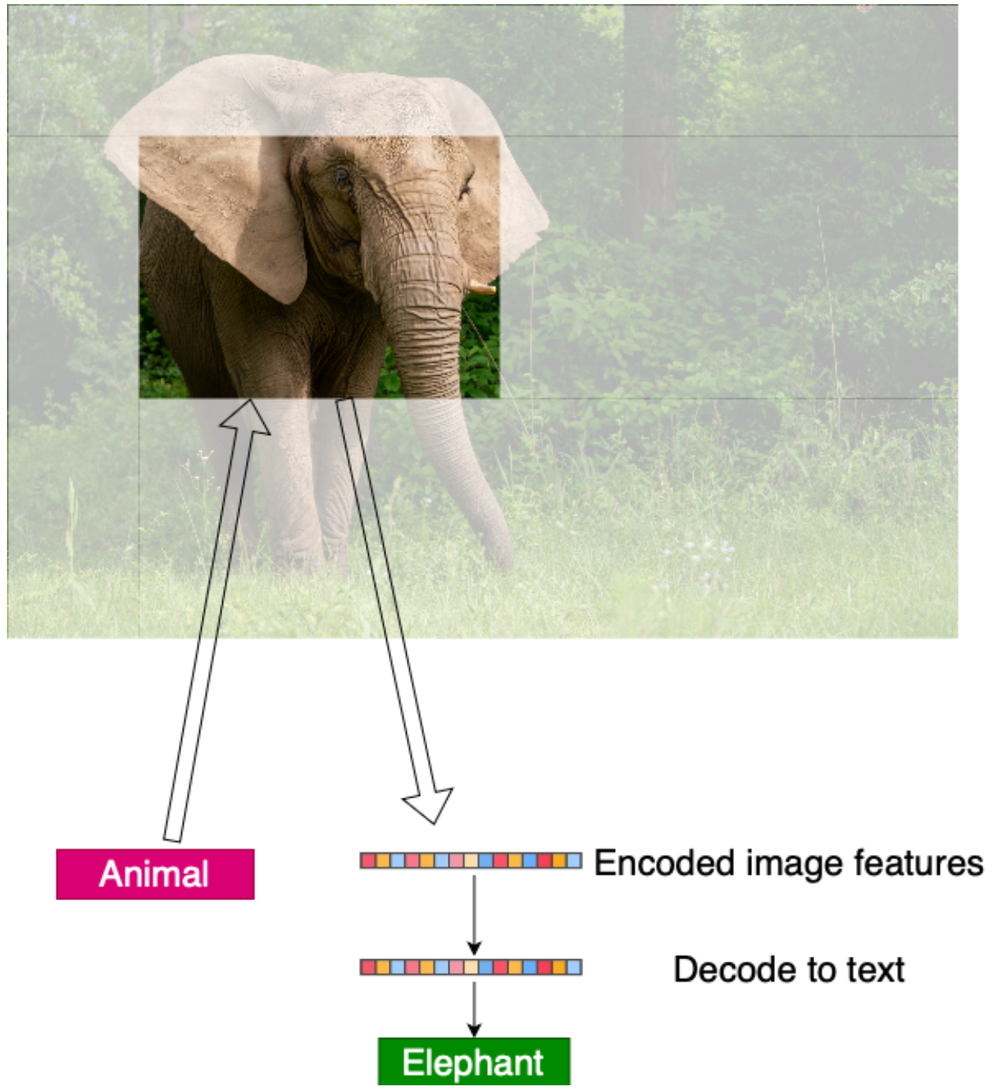
1. Separate encoding of question text and other modality
2. Combination of encoded features
3. Additional model layers to predict answer tokens

VQA

```
import requests
from PIL import Image

url = "https://www.worldanimalprotection
.org/cdn-cgi/image/width=1920,format=
auto/globalassets/images/elephants/1
033551-elephant.jpg"

image = Image.open(requests.get(url,
stream=True).raw)

text = "What animal is in this photo?"
```

VQA

- Model knows image and text features of many objects
- Reusable models with *no extra fine-tuning*



VQA

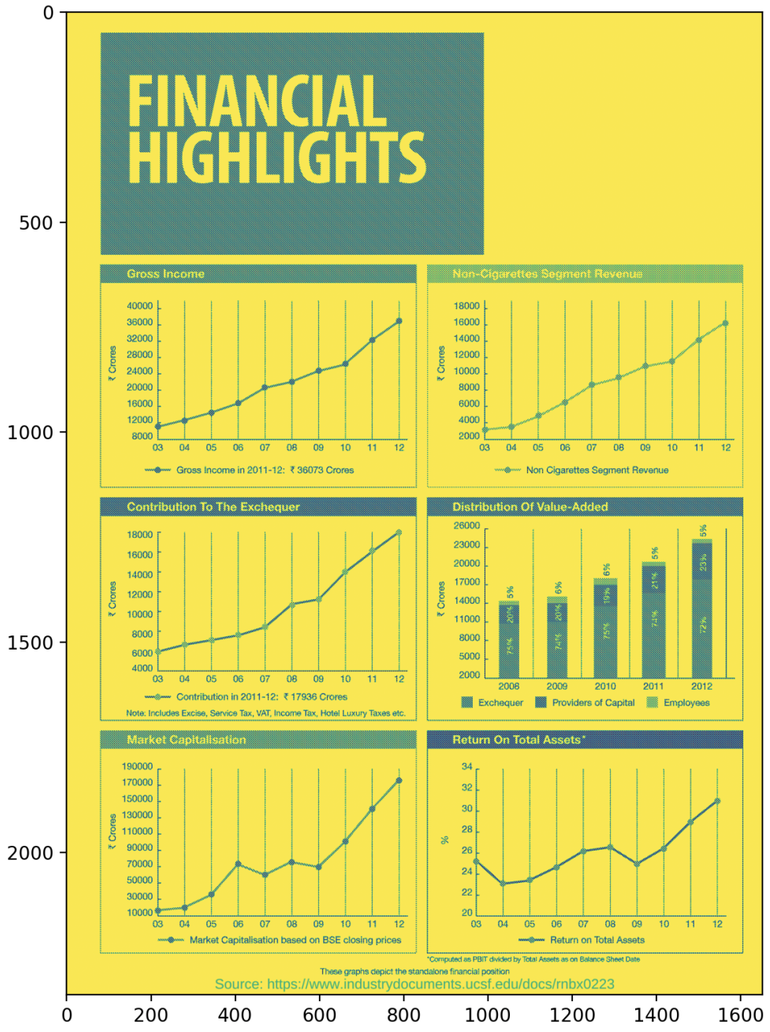
```
from transformers import ViltProcessor, ViltForQuestionAnswering

processor = ViltProcessor.from_pretrained("dandelin/vilt-b32-finetuned-vqa")
model = ViltForQuestionAnswering.from_pretrained("dandelin/vilt-b32-finetuned-vqa")

encoding = processor(image, text, return_tensors="pt")

outputs = model(**encoding)
idx = outputs.logits.argmax(-1).item()

print("Predicted answer:", model.config.id2label[idx])
```

```
Predicted answer: elephant
```

Document VQA

- Extension of VQA to detect graphs, tables, and text (**OCR**) in images

```
from datasets import load_dataset
from transformers import pipeline

dataset = load_dataset("lmms-lab/DocVQA")

import matplotlib.pyplot as plt
plt.imshow(dataset["test"][2]["image"])
plt.show()
```



Document VQA

Tesseract OCR

Tesseract

- Extra dependencies needed to run OCR
- `pytesseract` installed via `pip`
- Tesseract OCR via package **installer** (e.g. `apt-get`, `exe` or `homebrew` / `macports`)

Document VQA

- **LayoutLM**: Trained with images and Q/As from the **DocVQA dataset**

```
from transformers import pipeline
pipe = pipeline("document-question-answering", "impira/layoutlm-document-qa")

result = pipe(
    dataset["test"][2]["image"],
    "What was the gross income in 2011-2012?"
)
```

Document VQA

```
print(result)
```

```
[{'score': 0.05149758607149124,
  'answer': '3 36073 Crores', ...}]
```

Let's practice!

MULTI-MODAL MODELS WITH HUGGING FACE

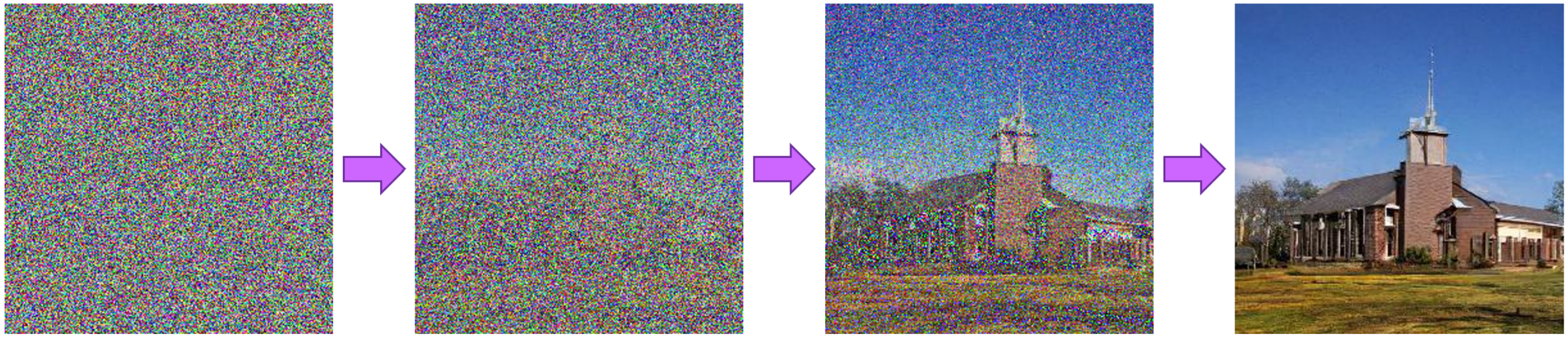
Image editing with diffusion models

MULTI-MODAL MODELS WITH HUGGING FACE

James Chapman
Curriculum Manager, DataCamp

datacamp

Diffusers

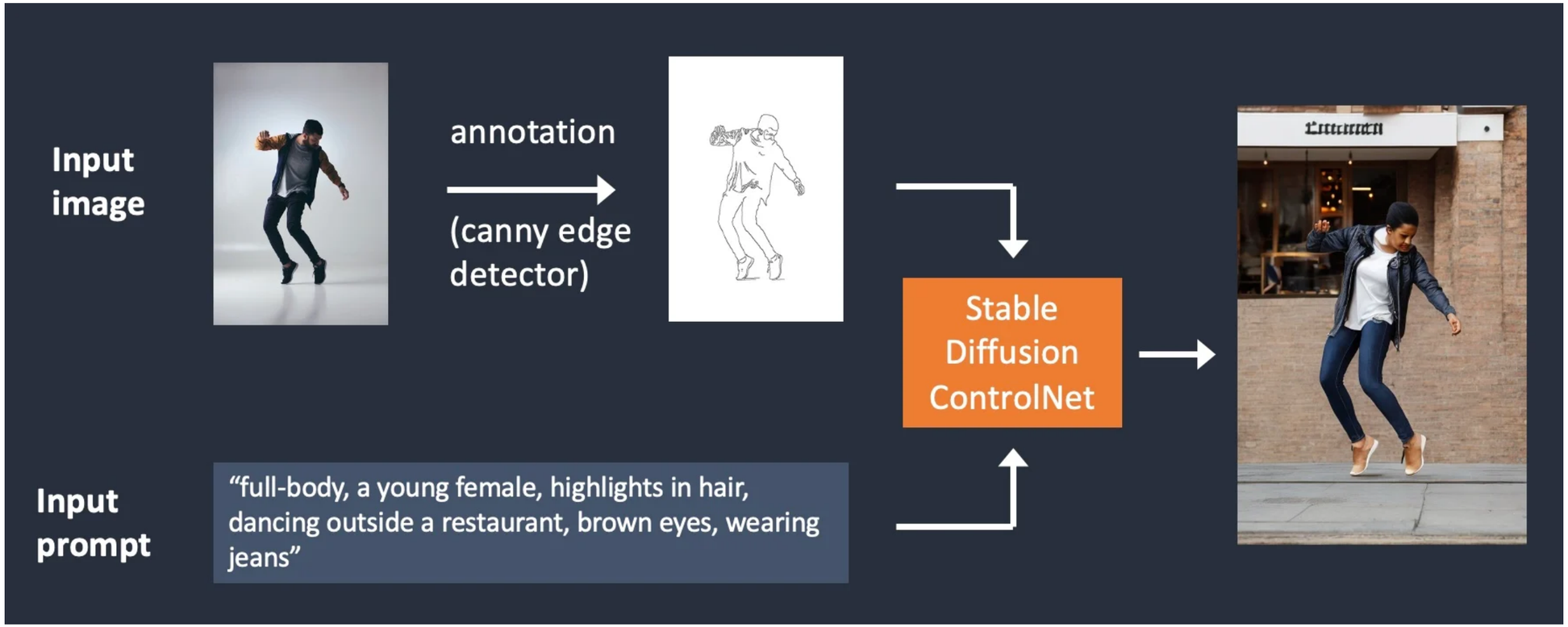


- Trained to map noise to an image
- CLIP + Diffusion → 2 types of conditional generation
  - Generation: Text → image
  - Modification: Text+image → image

[1] https://huggingface.co/docs/diffusers

Custom image editing

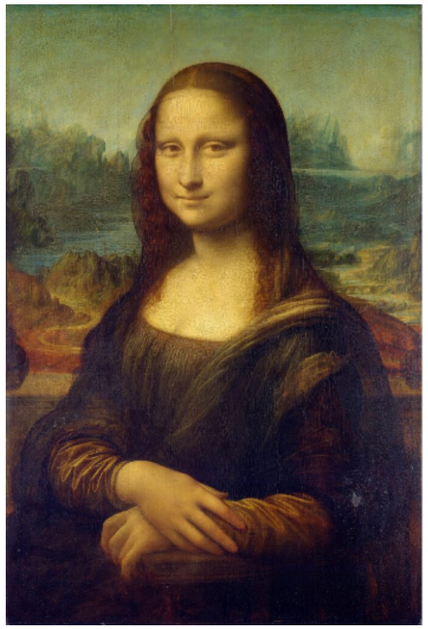
ControlNet: image and text-guided conditional generation



[1] https://stable-diffusion-art.com/controlnet/

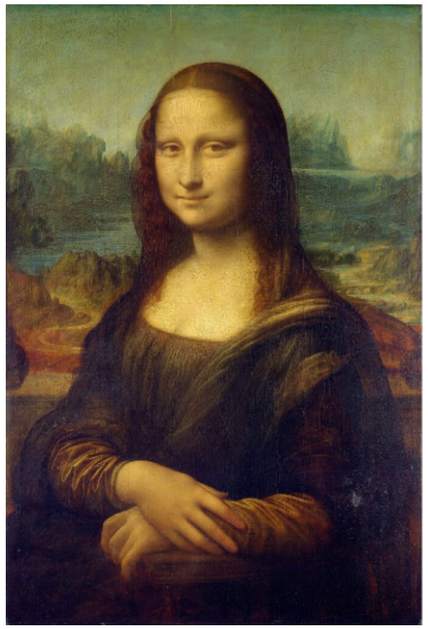
Custom image editing

```
from diffusers.utils import load_image

image = load_image("http://301.nz/o81bf")

import cv2
from PIL import Image
import numpy as np

image = cv2.Canny(np.array(image), 100, 200)
image = image[:, :, None]
image = np.concatenate([image, image, image],
                       axis=2)
```



Custom image editing

```
from diffusers.utils import load_image

image = load_image("http://301.nz/o81bf")

import cv2
from PIL import Image
import numpy as np

image = cv2.Canny(np.array(image), 100, 200)
image = image[:, :, None]
image = np.concatenate([image, image, image],
                       axis=2)
canny_image = Image.fromarray(image)
```



Custom image editing

```
from diffusers import StableDiffusionControlNetPipeline
from diffusers import ControlNetModel
import torch

controlnet = ControlNetModel.from_pretrained("lllyasviel/sd-controlnet-canny",
                                             torch_dtype=torch.float16)

pipe = StableDiffusionControlNetPipeline.from_pretrained(
    "runwayml/stable-diffusion-v1-5", controlnet=controlnet, torch_dtype=torch.float16
)

pipe = pipe.to("cuda")
```

Custom image editing

```
prompt = ["Albert Einstein,
          best quality,
          extremely detailed"]
generator = [
  torch.Generator(device="cuda").manual_seed(2)]

output = pipe(
    prompt,
    canny_image,
    negative_prompt=["monochrome,
                     lowres, bad anatomy,
                     worst quality,
                     low quality"],
    generator=generator,
    num_inference_steps=20)
```

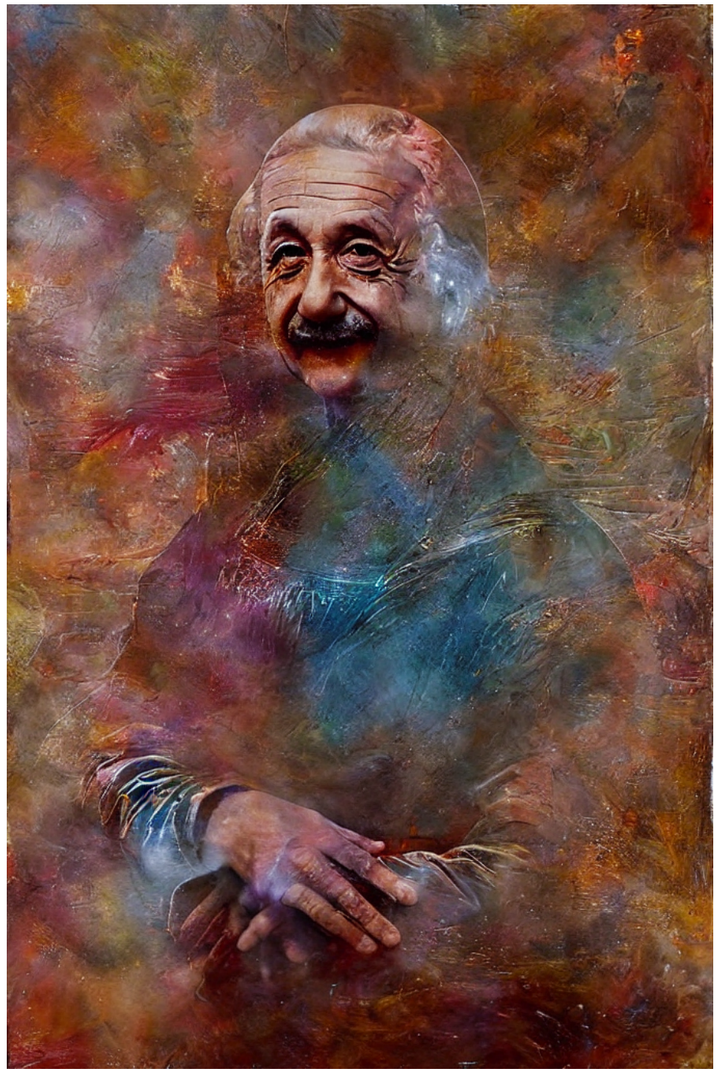


Image inpainting

- Generate new content *localized* to a certain region

Original image

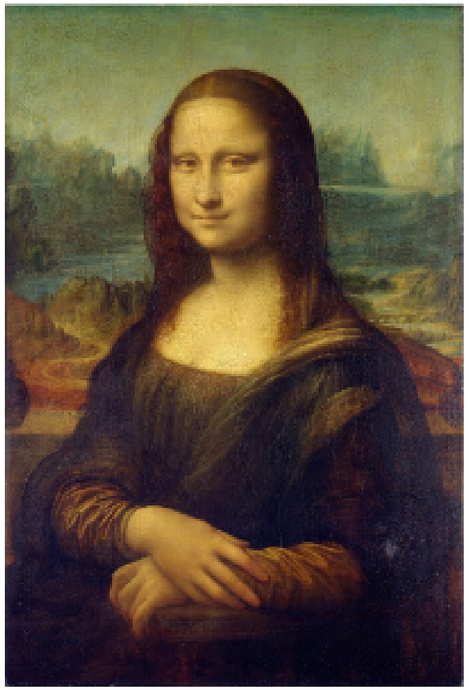
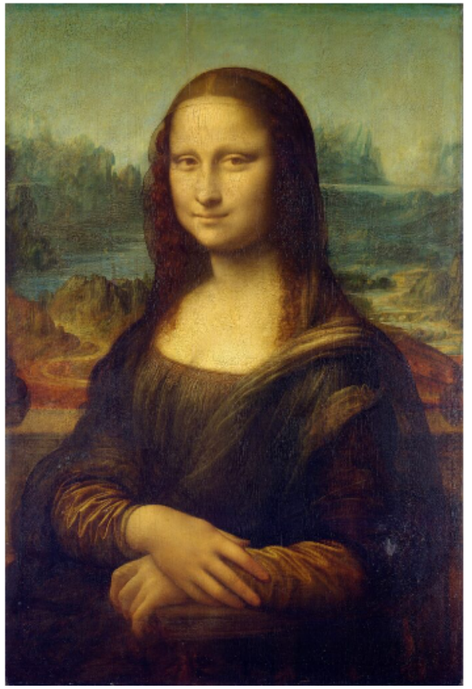


Image inpainting

- Generate new content *localized* to a certain region
- **Binary mask:** white (`1`), black (`0`)
- Masks from a segmentation or pre-defined by user (e.g., using **InpaintingMask-Generation**)

Original image



Corresponding mask

Image inpainting

```
from diffusers import StableDiffusionControlNetInpaintPipeline, ControlNetModel

controlnet = ControlNetModel.from_pretrained("lllyasviel/control_v11p_sd15_inpaint",
                                             torch_dtype=torch.float16,
                                             use_safetensors=True)

pipe = StableDiffusionControlNetInpaintPipeline.from_pretrained(
    "stable-diffusion-v1-5/stable-diffusion-v1-5",
    controlnet=controlnet, torch_dtype=torch.float16, use_safetensors=True
)
```

Image inpainting

```
def make_inpaint_condition(image, image_mask):
    image = np.array(image.convert("RGB")).astype(np.float32) / 255.0
    image_mask = np.array(image_mask.convert("L")).astype(np.float32) / 255.0

    image[image_mask > 0.5] = -1.0
    image = np.expand_dims(image, 0).transpose(0, 3, 1, 2)
    image = torch.from_numpy(image)
    return image

control_image = make_inpaint_condition(init_image, mask_image)
```

Image inpainting

```
output = pipe(
    "The head of the mona lisa in the
  same style and quality as the original
  mona lisa with a clear smile and a
  slightly smaller head size",
    num_inference_steps=40,
    eta=1.0,
    image=init_image,
    mask_image=mask_image,
    control_image=control_image,
).images[0]
```

Let's practice!

MULTI-MODAL MODELS WITH HUGGING FACE

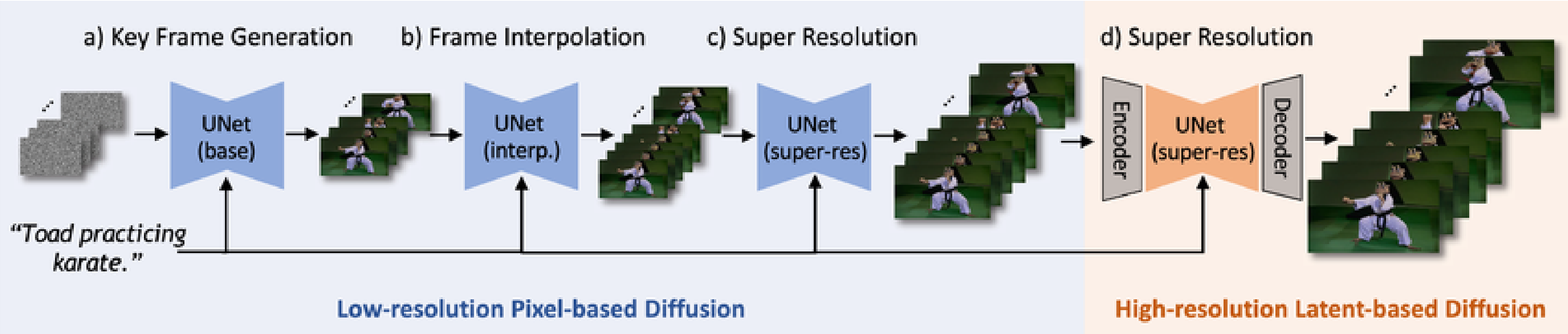
Video generation

MULTI-MODAL MODELS WITH HUGGING FACE

James Chapman
Curriculum Manager, DataCamp

datacamp

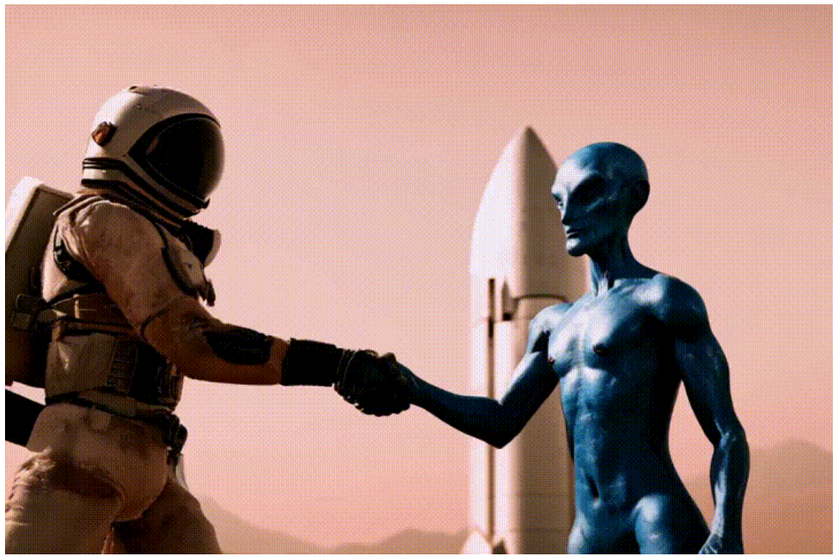
Video generation



[1] https://link.springer.com/article/10.1007/s11263-024-02271-9

Video generation

```
import torch
from diffusers import CogVideoXPipeline

pipe = CogVideoXPipeline.from_pretrained(
    "THUDM/CogVideoX-2b",
    torch_dtype=torch.float16
)

pipe.enable_model_cpu_offload()
pipe.enable_sequential__cpu_offload()
pipe.vae.enable_slicing()
pipe.vae.enable_tiling()
```



[1] https://huggingface.co/THUDM/CogVideoX-2b

Video generation

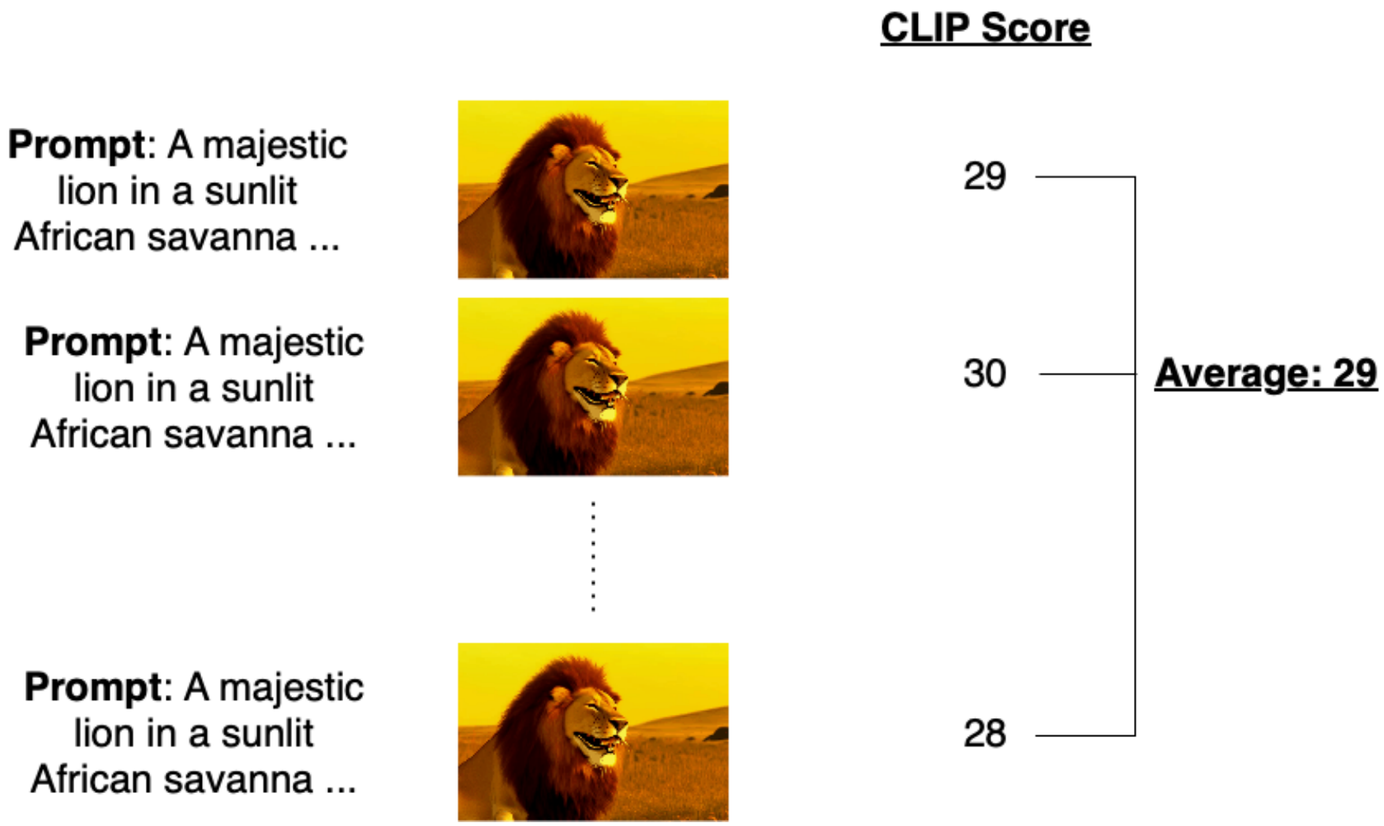
```
prompt = "A majestic lion in a sunlit African savanna, sitting regally
on a rock formation. Golden sunlight illuminates its magnificent mane,
then a big smile appears on its face"

video = pipe(
    prompt=prompt,
    num_inference_steps=20,
    num_frames=20,
    guidance_scale=6,
    generator=torch.Generator(device="cuda").manual_seed(42),
).frames[0]
```

Video generation

```
from diffusers.utils import export_to_video
from moviepy.editor import VideoFileClip

video_path = export_to_video(video,
                             "output.mp4",
                             fps=8)
video = VideoFileClip(video_path)
video.write_gif("video.gif")
```

Quantitative analysis

- Prompt adherence difficult for videos
- CLIP provides a possible strategy:



Quantitative analysis

```
from diffusers.utils import load_video
from torchmetrics.functional.multimodal import clip_score
from functools import partial
frames = load_video(video_path)
clip_score_fn = partial(clip_score, model_name_or_path="openai/clip-vit-base-patch16")
scores = []
for frame in frames:
  frame_int = np.array(frame).astype("uint8")
  frame_tensor = torch.from_numpy(frame_int).unsqueeze(0).permute(0, 3, 1, 2)
  score = clip_score_fn(frame_tensor, [prompt]).detach()
  scores.append(float(score))
avg_clip_score = round(np.mean(scores), 4)
print(f"Average CLIP score: {avg_clip_score}")
```

```
Average CLIP score: 30.6274
```

Let's practice!

MULTI-MODAL MODELS WITH HUGGING FACE

Congratulations!

MULTI-MODAL MODELS WITH HUGGING FACE

James Chapman
Curriculum Manager, DataCamp

Chapter 1

Text

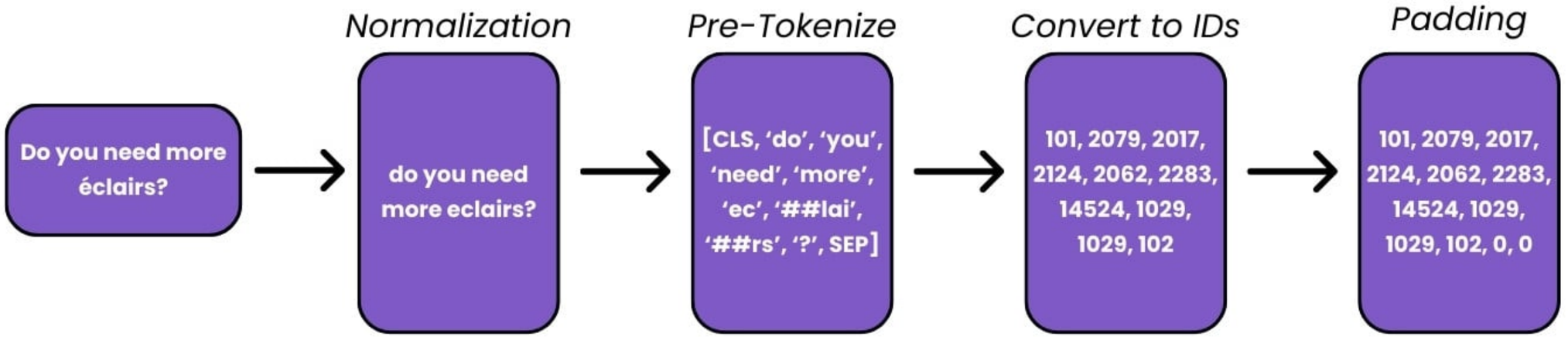
Here are five ways to learn about AI...

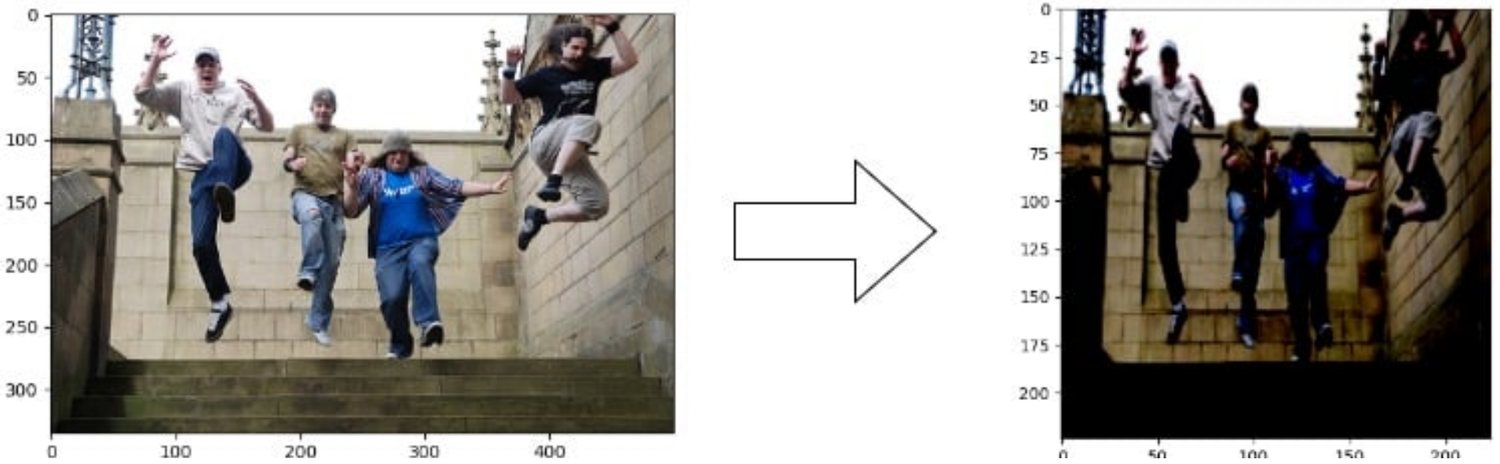
Image

Audio

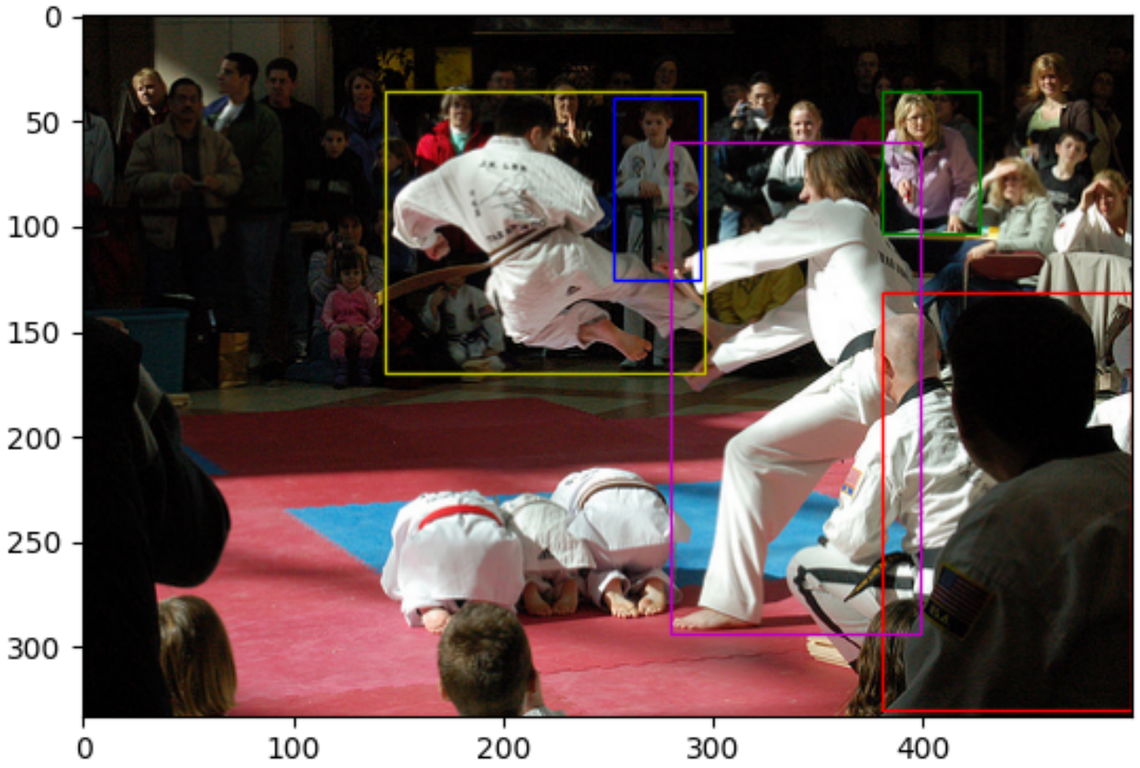
Video

Chapter 1

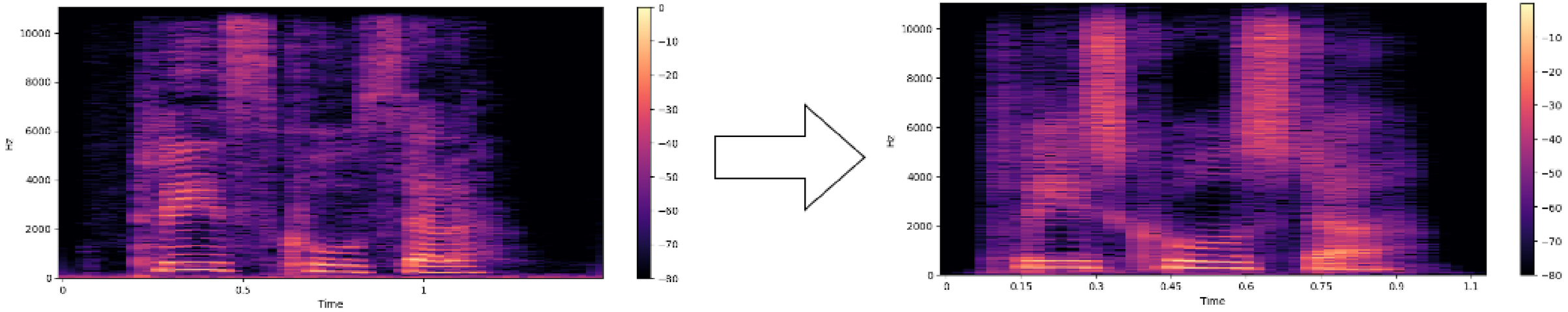

Normalization
Pre-Tokenize
Convert to IDs
Padding
Do you need more éclairs?
do you need more eclairs?
[CLS, 'do', 'you', 'need', 'more', 'ec', '##lai', '##rs', '?', SEP]
101, 2079, 2017, 2124, 2062, 2283, 14524, 1029, 1029, 102
101, 2079, 2017, 2124, 2062, 2283, 14524, 1029, 1029, 102, 0, 0



Chapter 2



Chapter 2

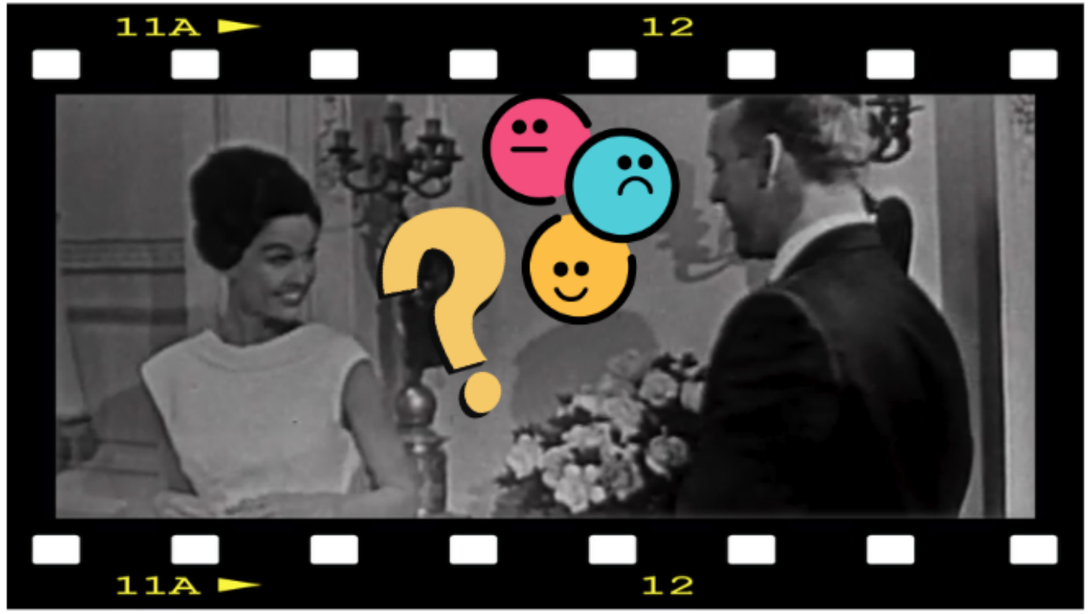

10000
8000
6000
4000
2000
0
Hz
0
0.5
1
Time
0
−10
−20
−30
−40
−50
−60
−70
−80
10000
8000
6000
4000
2000
0
Hz
0
0.15
0.3
0.45
0.6
0.75
0.9
1.1
Time
−10
−20
−30
−40
−50
−60
−70
−80

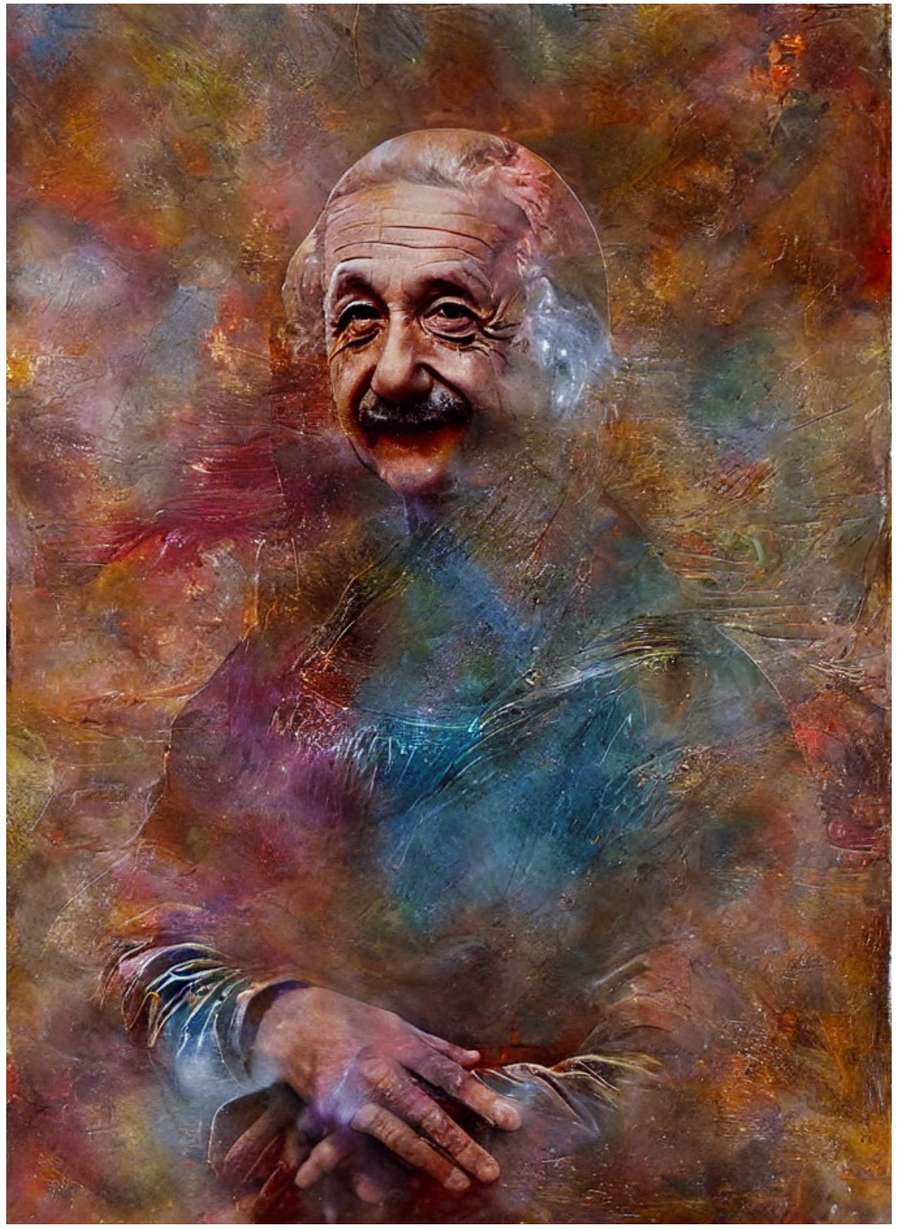
Chapter 3

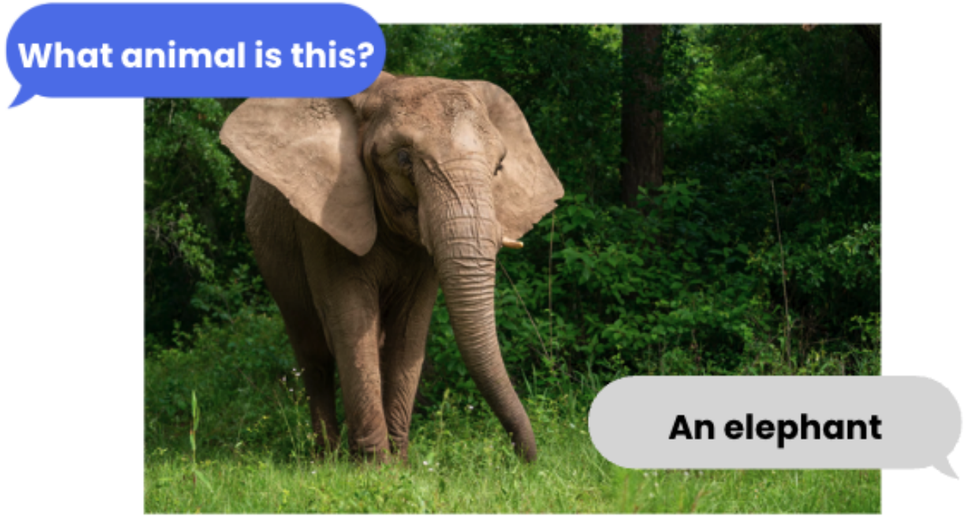
Macro Matters

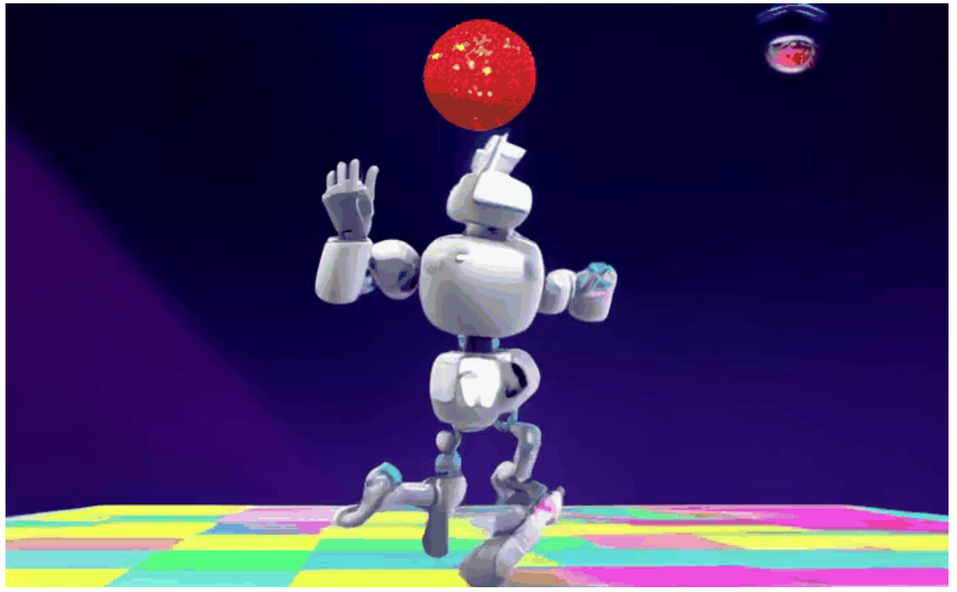
Japan's economy expands annualised 2.8% in Oct-Dec

1:05 AM GMT+1 · Updated 7 min ago

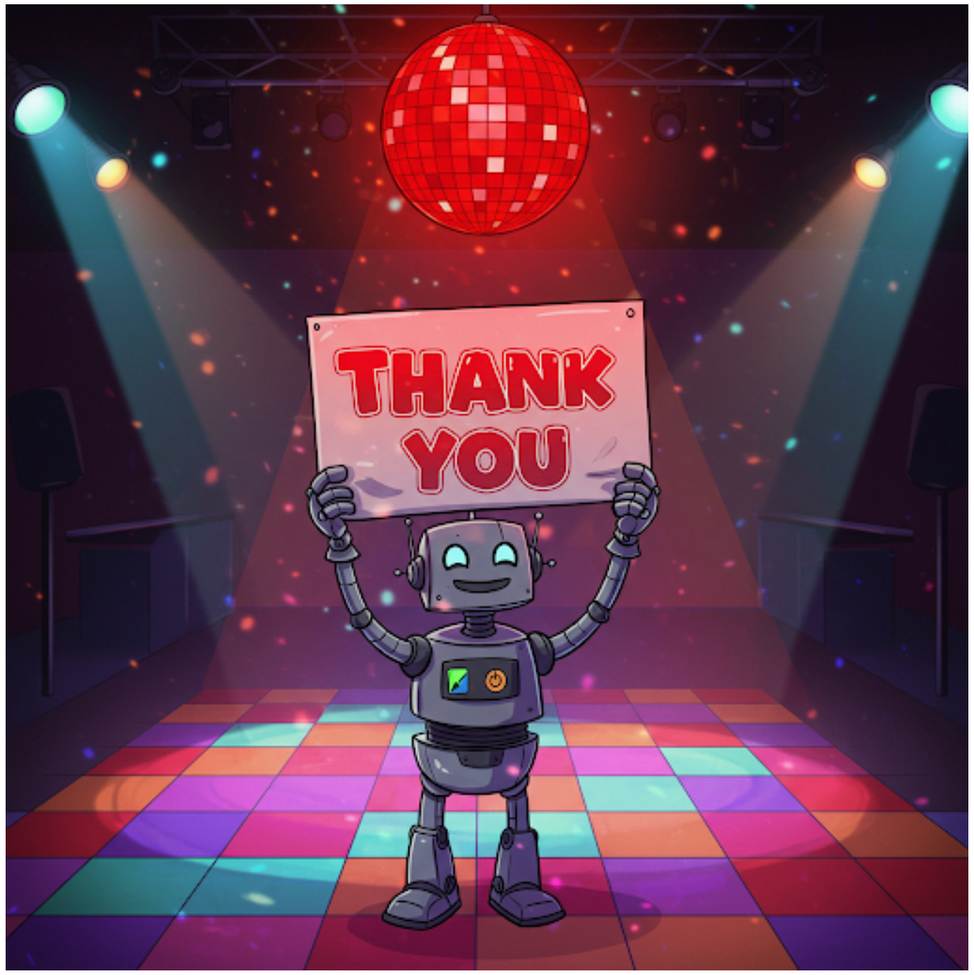





What animal is this?
An elephant



Bye and thanks!



Bye and thanks!

MULTI-MODAL MODELS WITH HUGGING FACE