
Foundation Model

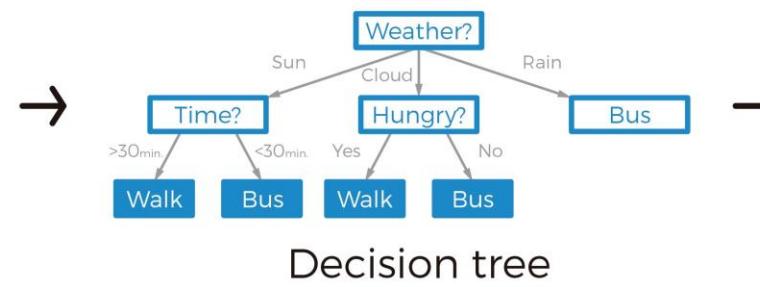
소프트웨어학과 김광수 교수 (kim.kwangsuh@skku.edu)
인공지능융합연구실 장승우 (sewo@skku.edu)

Background

Machine Learning



Input



Decision tree

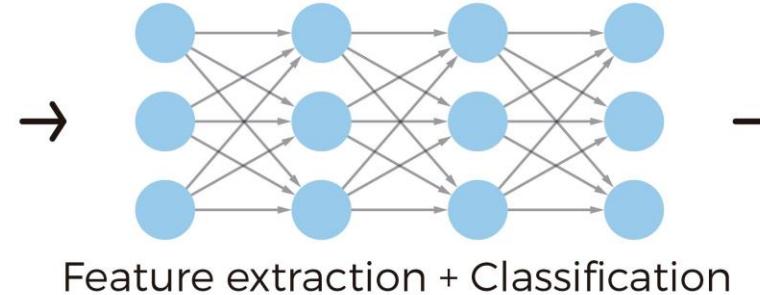


Output



Input

Deep Learning



Feature extraction + Classification



Output

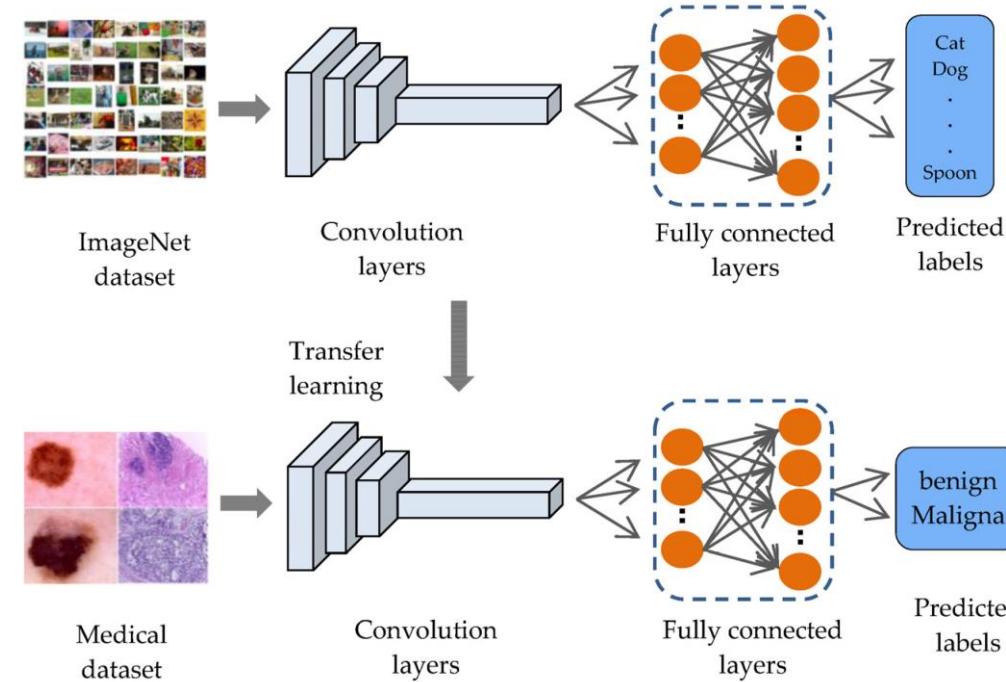
History of Deep Learning

- **Specialized DL** (Before BERT/Elmo in 2018)
 - Design specialized model architectures.
 - Leveraging task-specific features.
 - Train the specialized models with limited data.
- **Transfer DL** (Between 2018 - 2021)
 - Train a model with large amount of training data.
 - Use the features of the trained model to initialize part of the architecture
 - Design specialized modules on top of the trained features.
 - Train the partially specialized model with limited data.
- **Foundation Model** (After 2021)
 - Train a single huge model on astronomical amount of data
 - Prompt the single model for everything

Pros and Cons of Specialized DL

- Pros
 - The model considers the inductive bias for architecture design
 - The model can be effectively trained with limited amount of data
 - The model is normally small in size, easy to deploy for applications
- Cons
 - Each task requires lots of expertise for architecture design
 - Each task requires annotating specialized dataset
 - The model cannot benefit from other annotated data, it needs to start from scratch literally to gain its skill
 - Hosting many specialized models incur high costs

Transfer Learning

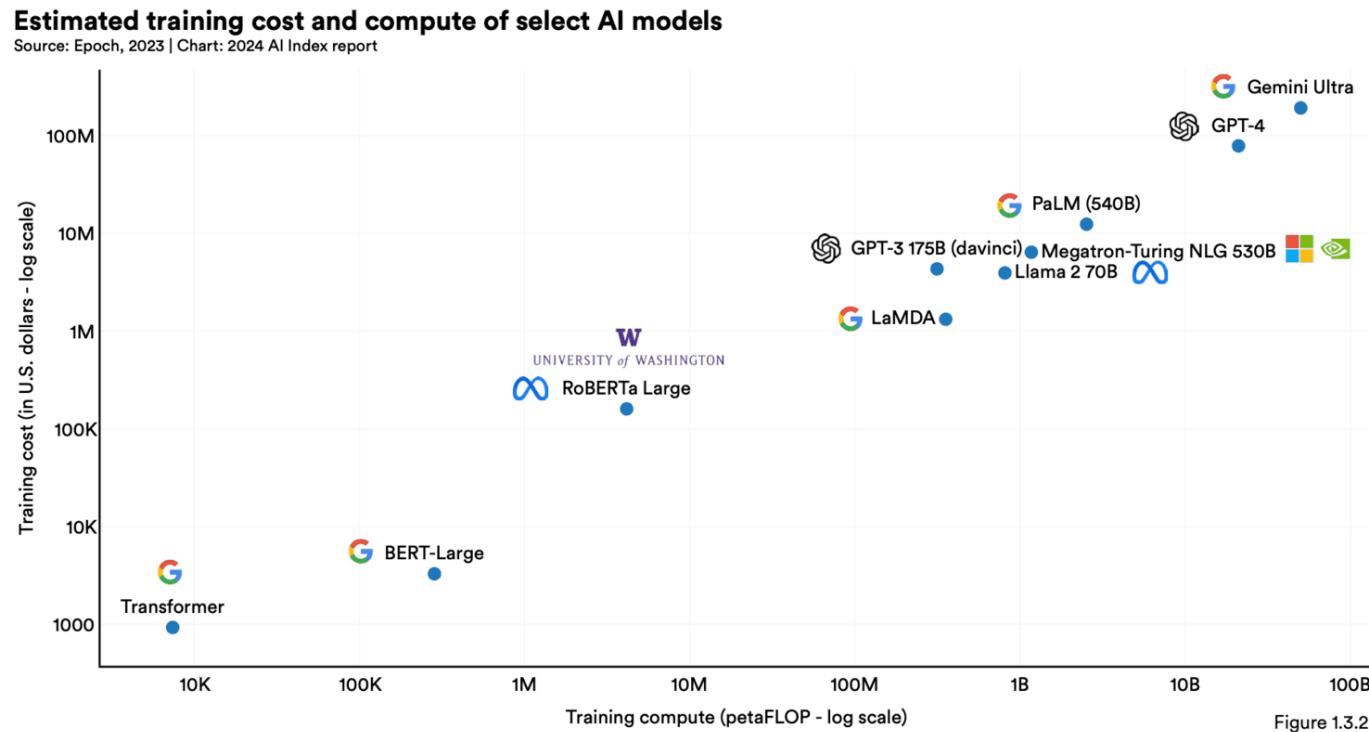


- We can train our model on massive amount of data to learn neural representation or initialize certain part of the weights.
- Then transfer to new tasks by adding layers on top of the learned neural representation.
- This can leverage some cross-task similarity to enhance model performance across different tasks.

Pros and Cons of Transfer DL

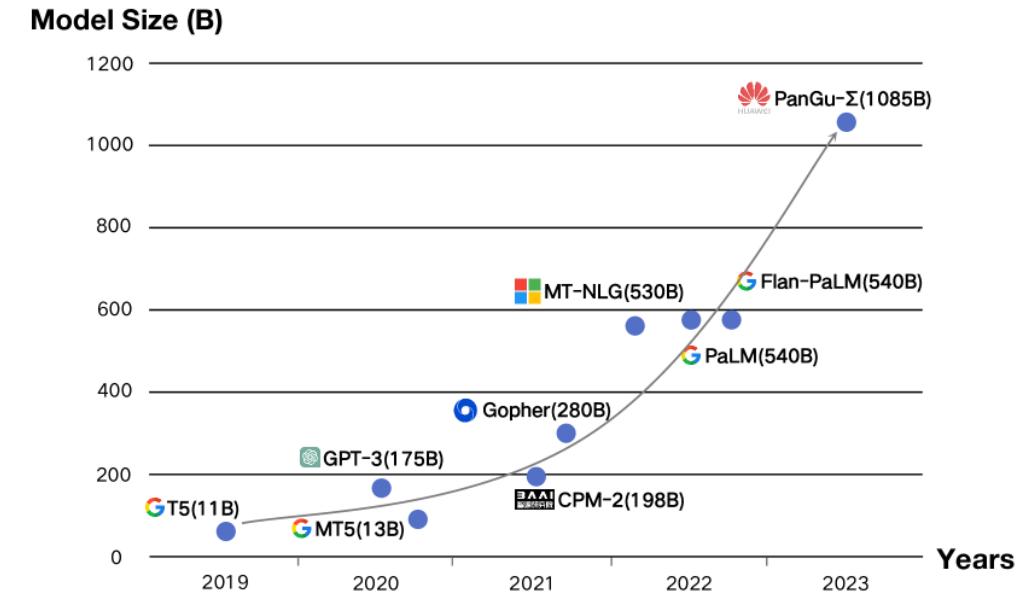
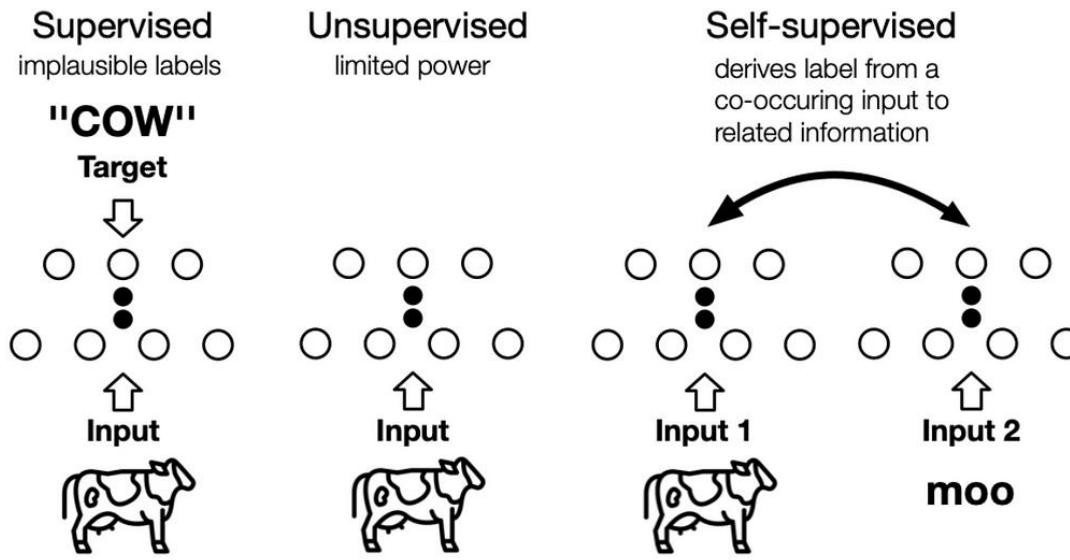
- Pros:
 - The model shows much stronger capability than Specialized DL
 - The model can generalize to unseen cases
 - The model requires very few fine-tuning
- Cons:
 - The model's performance is still not perfect.
 - There is still fine-tuning needed for the downstream tasks

Foundation Model



- A foundation model is a model trained at broad scale that can adapted to a wide range of downstream tasks.

Foundation Model

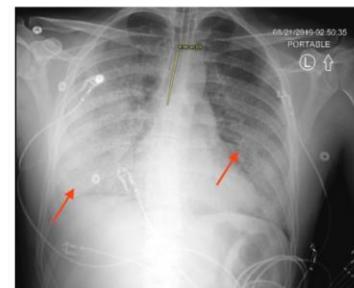
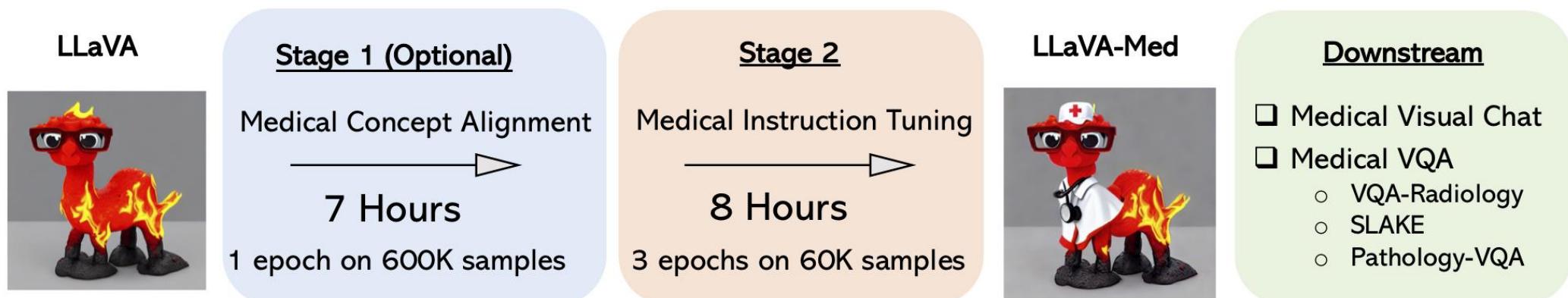


- The technology is not new:
Self-supervised learning with neural networks
- What is new?
Scale and the ability to perform task beyond training

Parameters

Parameters	Specialized DL	Transfer DL	Foundation DL
Model Size	< 100M parameters	100M -> 1B parameters	7B -> 1T parameters
Data Size	10K -> 1M tokens	100M -> 10B tokens	100B -> 30T tokens
Architecture	Specialized	General	General
Generalization	None	Reasonable	Strong

Applications



Caption: Chest x-ray (CXR) on Day 2 of admission post-intubation (yellow line showing the level of the endotracheal tube). Rapidly worsening ground-glass opacities are seen throughout the lungs with relative subpleural sparing (red arrows)

Source: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7398722/>

User
LLaVA-Med

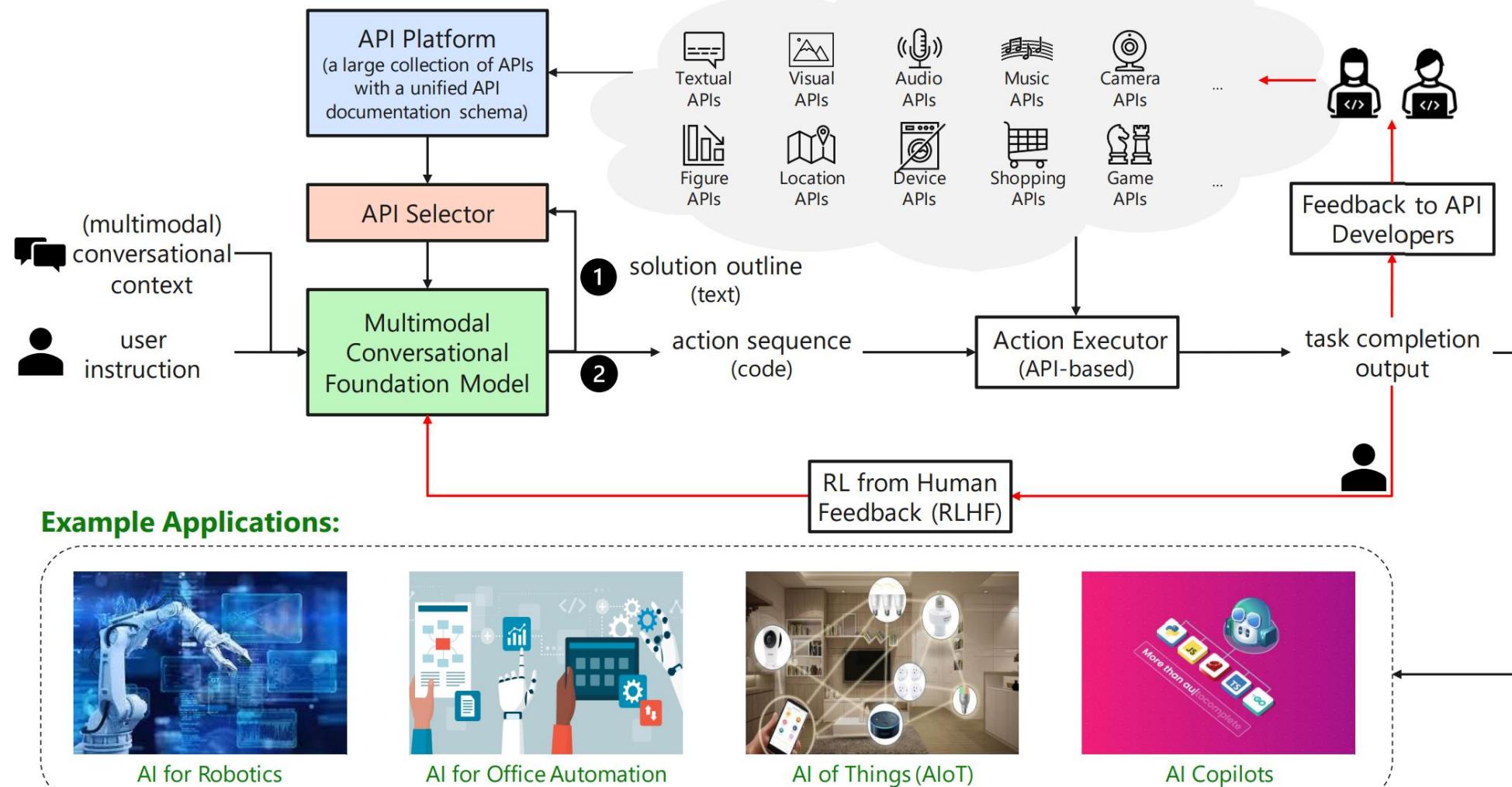
What is shown in this image?
The image is a chest X-ray (CXR) that shows bilateral patchy infiltrates, which are areas of increased opacity in the lungs. These infiltrates can be indicative of various lung conditions, such as infections, inflammation, or other lung diseases.

User
LLaVA-Med

Are there any devices implanted in this patient?
Yes, it appears that the patient has several devices implanted. These devices may include endotracheal tubes, central venous catheters, and/or pulmonary artery catheters. These devices are typically used for monitoring and managing various medical conditions, as well as for providing support during surgical procedures.

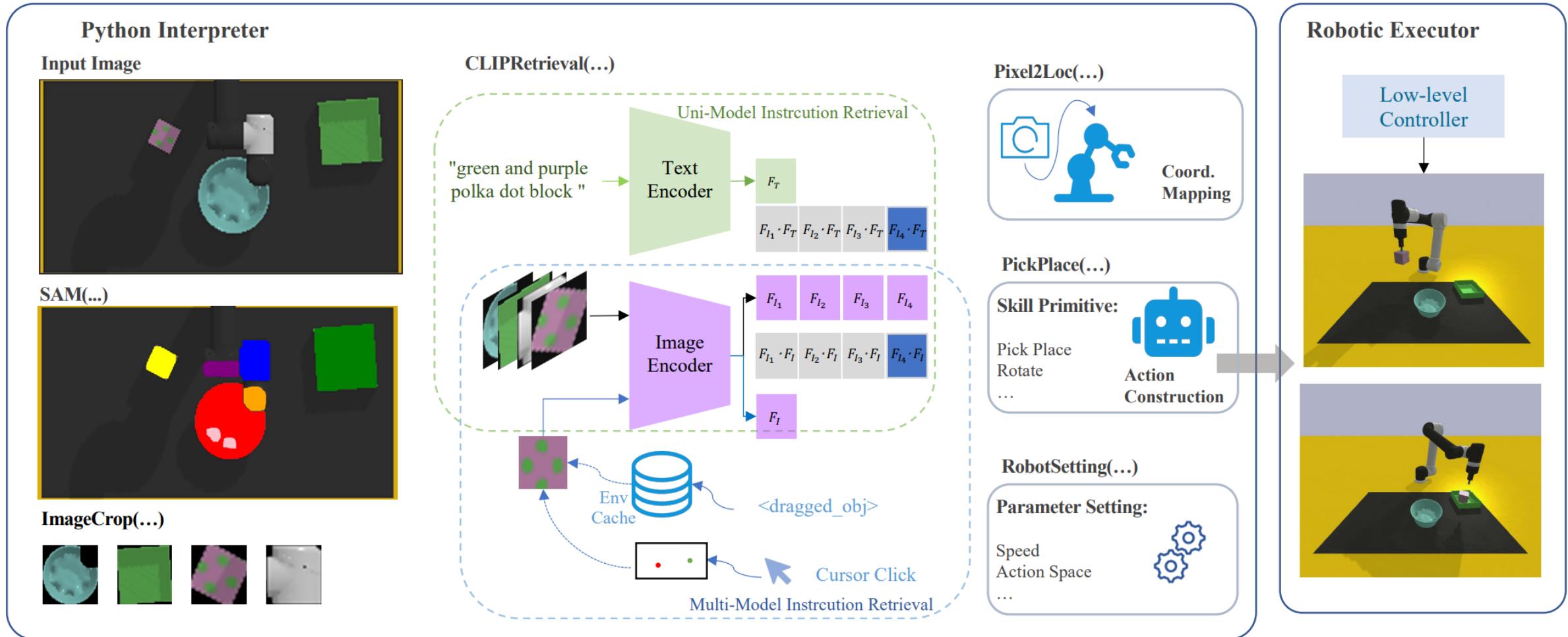
Applications

TaskMatrix.AI: Completing Tasks by Connecting Foundation Models with **Millions of APIs**



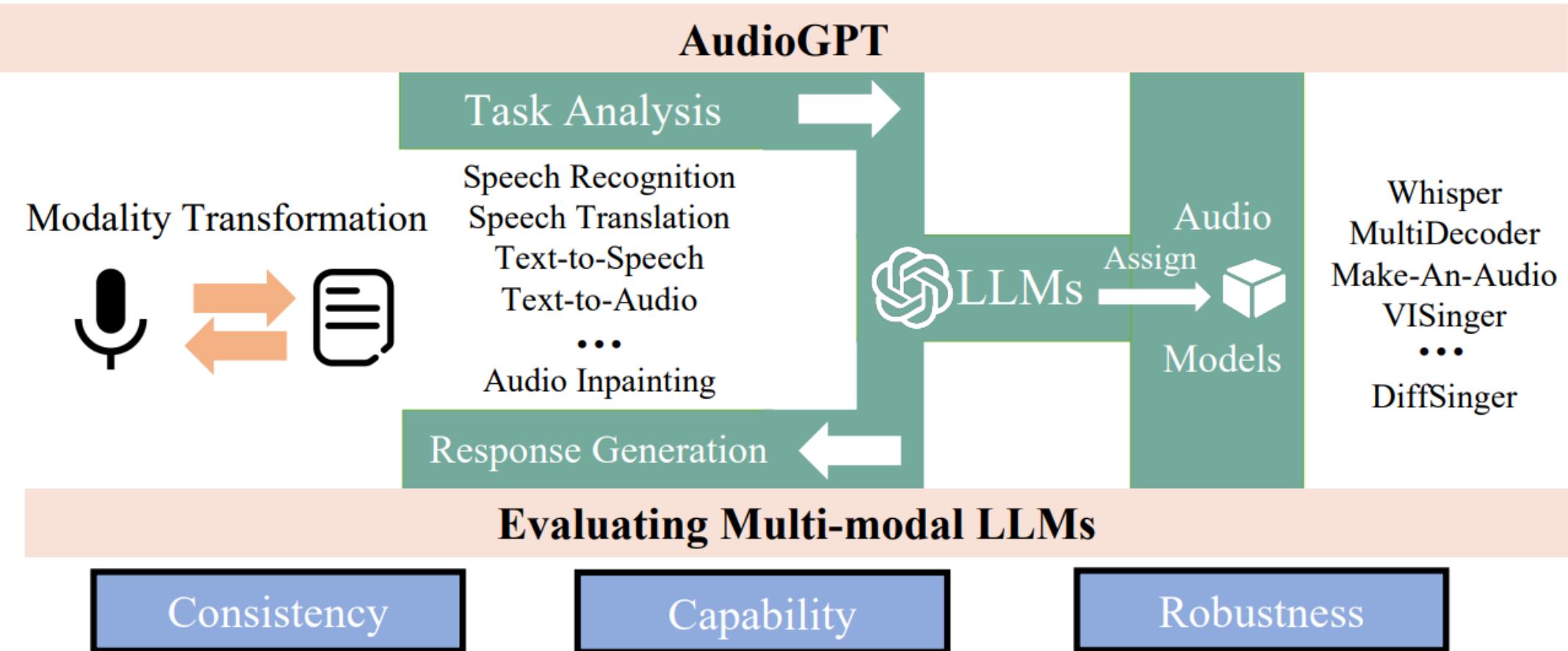
Applications

Instruct2Act: Mapping Multi-modality Instructions to Robotic Actions with Large Language Model



Applications

AudioGPT: Understanding and Generating Speech, Music, Sound, and Talking Head



Hugging Face

What is Hugging Face?



A major hub for open-source machine learning

- Collaboration platform
- Open-source machine learning
- Text, vision, and audio tasks
- Models, datasets, frameworks

What is Hugging Face Hub?

The screenshot shows the Hugging Face Hub's main interface. At the top, there's a navigation bar with tabs for Tasks, Libraries, Datasets, Languages, Licenses, and Other. Below this is a search bar with a placeholder 'Filter Tasks by name'. The main content area is divided into sections for Multimodal and Computer Vision tasks.

Multimodal Tasks:

- Image-Text-to-Text
- Visual Question Answering
- Document Question Answering

Computer Vision Tasks:

- Depth Estimation
- Image Classification
- Object Detection
- Image Segmentation
- Text-to-Image
- Image-to-Text
- Image-to-Image
- Image-to-Video
- Unconditional Image Generation
- Video Classification
- Text-to-Video
- Zero-Shot Image Classification
- Mask Generation
- Zero-Shot Object Detection
- Text-to-3D
- Image-to-3D
- Image Feature Extraction

Models List:

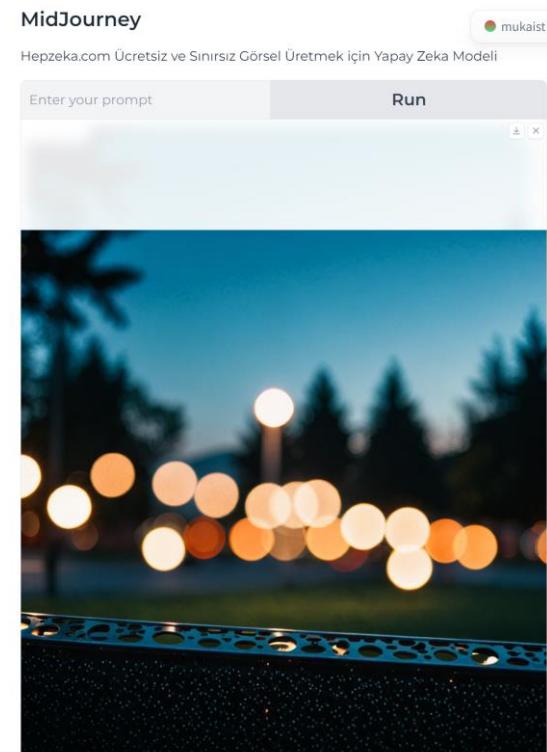
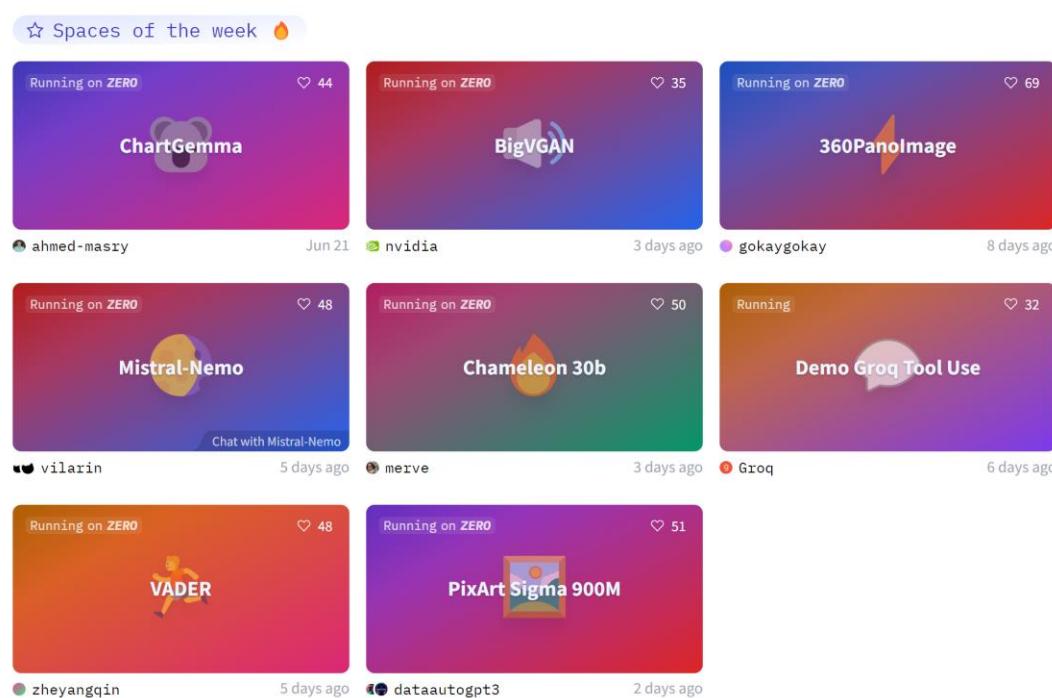
- mistralai/Mistral-Nemo-Instruct-2407** (Text Generation, Updated about 18 hours ago, 60.7k, 737)
- apple/DCLM-7B** (Text Generation, Updated 4 days ago, 2.11k, 605)
- meta-llama/Meta-Llama-3.1-8B-Instruct** (Text Generation, Updated 1 day ago, 84.4k, 684)
- mistralai/Mistral-Large-Instruct-2407** (Text Generation, Updated about 13 hours ago, 357)
- meta-llama/Meta-Llama-3.1-405B-Instruct** (Text Generation, Updated 1 day ago, 3.14k, 280)
- mistralai/Mistral-Nemo-Base-2407** (Text Generation, Updated about 18 hours ago, 6.02k, 199)
- xinsir/controlnet-union-sdxl-1.0** (Text-to-Image, Updated 4 days ago, 41.2k, 749)
- stabilityai/stable-diffusion-3-medium** (Text-to-Image, Updated 13 days ago, 166k, 3.66k)
- mistralai/mamba-codestral-7B-v0.1** (Updated about 18 hours ago, 3, 407)

- The Hugging Face Hub is a collaboration platform that hosts a huge collection of open-source models and datasets for machine learning, think of its being like Github for ML.

HuggingFace Datasets

- **HuggingFace Datasets** is a lightweight library providing two main features:
 - **One-line data loaders for many public dataset**: one-liners to download and pre-process any of the number of datasets major public datasets.
 - With a simple command like `squad_dataset = load_dataset('squad')`, get any of these datasets ready
 - **Efficient data pre-processing**: simple, fast and reproducible data pre-processing for the public datasets as well as your own local datasets in CSV, JSON, text, PNG, JPEG, WAV, MP3, Parquet, etc.
 - With simple commands like `processed_dataset = dataset.map(process_example)`

What is Hugging Face Space?



- Spaces from Hugging Face is a service available on the Hugging Face Hub that provides an easy to use GUI for building and deploying web hosted ML demos and apps.

HuggingFace Transformers

- **Transformers** provides thousands of pretrained models to perform tasks on different modalities such as text, vision, and audio.
- These models can be applied on:

Text, for tasks like text classification, information extraction question answering, summarization, translation, and text generation, in over 100 languages.

Images, for tasks like image classification, object detection, and segmentation.

Audio, for tasks like speech recognition and audio classification.

How to using Transformers?

```
from transformers import ViTImageProcessor, ViTForImageClassification  
from PIL import Image  
import requests
```

Load image

```
url = 'http://images.cocodataset.org/val2017/000000039769.jpg'  
image = Image.open(requests.get(url, stream=True).raw)
```

Load pre-defined preprocessor
and pre-trained model

```
processor = ViTImageProcessor.from_pretrained('google/vit-base-patch16-224')  
model = ViTForImageClassification.from_pretrained('google/vit-base-patch16-224')
```

```
inputs = processor(images=image, return_tensors="pt")  
outputs = model(**inputs)  
logits = outputs.logits  
# model predicts one of the 1000 ImageNet classes  
predicted_class_idx = logits.argmax(-1).item()  
print("Predicted class:", model.config.id2label[predicted_class_idx])
```

Prediction

HuggingFace vision ecosystem

Set-up environment

Install the required libraries: 😊 Transformers and 😊 Datasets

```
pip install transformers datasets
```

Models

👏 Transformers currently includes a large amount of vision models for various tasks.

First load a Vision Transformer (ViT)

Instantiating a model without pre-trained weights can be done by

- 1) instantiating a configuration, defining the model architecture
- 2) Creating a model based on that configuration

```
from transformers import ViTConfig, ViTForImageClassification  
  
config = ViTConfig(num_hidden_layers=12, hidden_size=768)  
model = ViTForImageClassification(config)
```

Models

The configuration just stores the hyperparameters related to the architecture of the model.

```
print(config)

ViTConfig {
    "attention_probs_dropout_prob": 0.0,
    "encoder_stride": 16,
    "hidden_act": "gelu",
    "hidden_dropout_prob": 0.0,
    "hidden_size": 768,
    "image_size": 224,
    "initializer_range": 0.02,
    "intermediate_size": 3072,
    "layer_norm_eps": 1e-12,
    "model_type": "vit",
    "num_attention_heads": 12,
    "num_channels": 3,
    "num_hidden_layers": 12,
    "patch_size": 16,
    "qkv_bias": true,
    "transformers_version": "4.43.2"
}
```

Load a model

- Alternatively, is to equip a model with pre-trained weights, such that it can easily fine-tuned on a custom dataset.

```
model = ViTForImageClassification.from_pretrained("google/vit-base-patch16-224")
```

config.json: 100%  69.7k/69.7k [00:00<00:00, 409kB/s]

model.safetensors: 100%  346M/346M [00:03<00:00, 104MB/s]

- Note that any model (and any dataset, Space) on the hub has its own Git repo with its own Git history.

```
model = ViTForImageClassification.from_pretrained("google/vit-base-patch16-224",
                                                revision="db75733ce9ead4ed3dce26ab87a6ed2f6f565985")
```

pytorch_model.bin: 100%  346M/346M [00:03<00:00, 95.6MB/s]

Load a feature extractor

- A feature extractor can be used to prepare images for the model.
- It's a minimal object to prepare images for inference.
- It typically does some very simple image transformations (like resizing to fixed size + normalizing).

```
from transformers import ViTImageProcessor  
  
feature_extractor = ViTImageProcessor()  
feature_extractor = ViTImageProcessor.from_pretrained("google/vit-base-patch16-224")
```

Predict on image

- Load a cats image of the COCO dataset

```
from PIL import Image
import requests

url = 'http://images.cocodataset.org/val2017/000000039769.jpg'
image = Image.open(requests.get(url, stream=True).raw)
image.save("cats.png")
image
```



- Prepare for the model

```
inputs = feature_extractor(image, return_tensors="pt")
pixel_values = inputs.pixel_values
print(pixel_values.shape)

torch.Size([1, 3, 224, 224])
```

Pipeline

- The Pipeline API allows you to just feed a text/image/audio to it and get a prediction out. All complexity is handled by the pipeline.

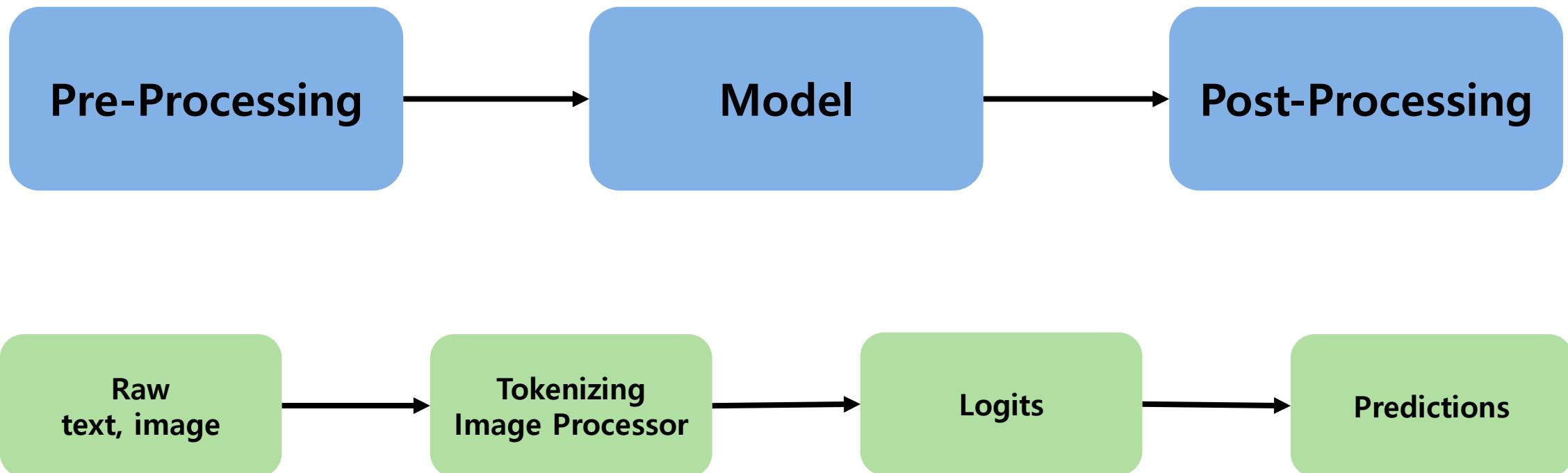


Image classification pipeline

```
from transformers import pipeline
image_path = "https://farm5.staticflickr.com/4007/4322154488_997e69e4cf_z.jpg"
pipe = pipeline("image-segmentation", model="briaai/RMBG-1.4", trust_remote_code=True)
pillow_mask = pipe(image_path, return_mask = True) # outputs a pillow mask
pillow_image = pipe(image_path) # applies mask on input and returns a pillow image
```

```
from transformers import AutoModelForImageSegmentation
from torchvision.transforms.functional import normalize
model = AutoModelForImageSegmentation.from_pretrained("briaai/RMBG-1.4", trust_remote_code=True)
def preprocess_image(im: np.ndarray, model_input_size: list) -> torch.Tensor:
    if len(im.shape) < 3:
        im = im[:, :, np.newaxis]
    # orig_im_size=im.shape[0:2]
    im_tensor = torch.tensor(im, dtype=torch.float32).permute(2,0,1)
    im_tensor = F.interpolate(torch.unsqueeze(im_tensor,0), size=model_input_size)
    image = torch.divide(im_tensor,255.0)
    image = normalize(image,[0.5,0.5,0.5],[1.0,1.0,1.0])
    return image

def postprocess_image(result: torch.Tensor, im_size: list)-> np.ndarray:
    result = torch.squeeze(F.interpolate(result, size=im_size, mode='bilinear'))
    ma = torch.max(result)
    mi = torch.min(result)
    result = (result-mi)/(ma-mi)
    im_array = (result*255).permute(1,2,0).cpu().data.numpy().astype(np.uint8)
    im_array = np.squeeze(im_array)
    return im_array

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model.to(device)

# prepare input
image_path = "https://farm5.staticflickr.com/4007/4322154488_997e69e4cf_z.jpg"
orig_im = io.imread(image_path)
orig_im_size = orig_im.shape[0:2]
image = preprocess_image(orig_im, model_input_size).to(device)

# inference
result=model(image)

# post process
result_image = postprocess_image(result[0][0], orig_im_size)

# save result
pil_im = Image.fromarray(result_image)
no_bg_image = Image.new("RGBA", pil_im.size, (0,0,0,0))
orig_image = Image.open(image_path)
no_bg_image.paste(orig_image, mask=pil_im)
```

Pipeline API

- Image classification is probably the simplest vision task: given an image, predict which class(es) belong to it.

```
from transformers import pipeline
image_pipe = pipeline("image-classification")
```

No model was supplied, defaulted to google/vit-base-patch16-224 and revision 5dca96d (<https://huggingface.co/google/vit-base-patch16-224>).
Using a pipeline without specifying a model name and revision in production is not recommended.
Fast image processor class <class 'transformers.models.vit.image_processing_vit_fast.VITImageProcessorFast'> is available for this model. Using slow image processor classes. To use the fast image processor class set `use_fast=True`.
Hardware accelerator e.g. GPU is available in the environment, but no `device` argument is passed to the `Pipeline` object. Model will be on CPU.

```
image_pipe(image)
[{'label': 'Egyptian cat', 'score': 0.9374414682388306},
 {'label': 'tabby, tabby cat', 'score': 0.03844264894723892},
 {'label': 'tiger cat', 'score': 0.01441141590476036},
 {'label': 'lynx, catamount', 'score': 0.003274328541010618},
 {'label': 'Siamese cat, Siamese', 'score': 0.0006795949884690344}]
```

Pipeline API

- Note that you can also provide a custom model from the hub:

```
image_pipe = pipeline("image-classification",
                      model="microsoft/swin-tiny-patch4-window7-224")
```

```
config.json: 100% [██████████] 71.8k/71.8k [00:00<00:00, 1.07MB/s]
```

```
model.safetensors: 100% [██████████] 113M/113M [00:01<00:00, 95.1MB/s]
```

```
preprocessor_config.json: 100% [██████████] 255/255 [00:00<00:00, 28.6kB/s]
```

Hardware accelerator e.g. GPU is available in the environment, but no `device` argument is passed to the `Pipeline` object. Model will be on CPU.

```
image_pipe(image)
```

```
[{'label': 'tabby, tabby cat', 'score': 0.3746733069419861},
 {'label': 'tiger cat', 'score': 0.14187975227832794},
 {'label': 'Egyptian cat', 'score': 0.11662543565034866},
 {'label': 'remote control, remote', 'score': 0.059291254729032516},
 {'label': 'crutch', 'score': 0.017978264018893242}]
```

Pipeline API

- model + feature extractor

```
feature_extractor = AutoFeatureExtractor.from_pretrained("facebook/convnext-tiny-224")
model = AutoModelForImageClassification.from_pretrained("facebook/convnext-tiny-224")

image_pipe = pipeline("image-classification",
                      model=model,
                      feature_extractor=feature_extractor)
```

Hardware accelerator e.g. GPU is available in the environment, but no `device` argument is passed to the `Pipeline` object. Model will be on CPU.

```
image_pipe(image)

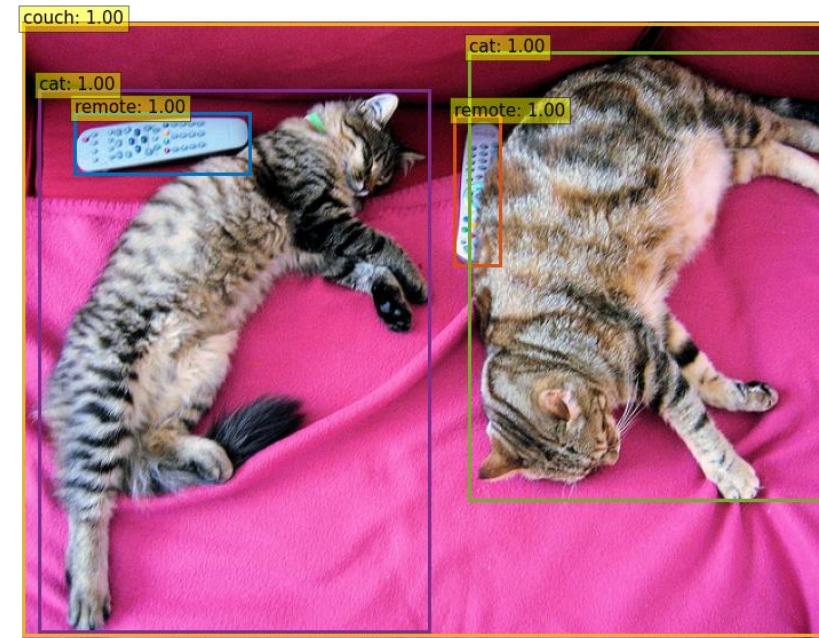
[{'label': 'tabby, tabby cat', 'score': 0.613274872303009},
 {'label': 'tiger cat', 'score': 0.2736639380455017},
 {'label': 'Egyptian cat', 'score': 0.009575308300554752},
 {'label': 'quilt, comforter, comfort, puff', 'score': 0.001771230949088931},
 {'label': 'pillow', 'score': 0.0017434763722121716}]
```

Object detection pipeline

```
object_detection_pipe = pipeline("object-detection",
                                model="facebook/detr-resnet-50")
```

```
results = object_detection_pipe(image)
results

[{'score': 0.9982202649116516,
 'label': 'remote',
 'box': {'xmin': 40, 'ymin': 70, 'xmax': 175, 'ymax': 117}},
 {'score': 0.9960021376609802,
 'label': 'remote',
 'box': {'xmin': 333, 'ymin': 72, 'xmax': 368, 'ymax': 187}},
 {'score': 0.9954743981361389,
 'label': 'couch',
 'box': {'xmin': 0, 'ymin': 1, 'xmax': 639, 'ymax': 473}},
 {'score': 0.99880051612854,
 'label': 'cat',
 'box': {'xmin': 13, 'ymin': 52, 'xmax': 314, 'ymax': 470}},
 {'score': 0.9986783862113953,
 'label': 'cat',
 'box': {'xmin': 345, 'ymin': 23, 'xmax': 640, 'ymax': 368}}]
```



Depth estimation pipeline

```
depth_estimation_pipe = pipeline("depth-estimation")
```

```
results = depth_estimation_pipe(image)
results

{'predicted_depth': tensor([[[ 6.3199,  6.3628,  6.4148, ... , 10.4104, 10.5109, 10.3847],
    [ 6.3850,  6.3615,  6.4166, ... , 10.4540, 10.4384, 10.4554],
    [ 6.3519,  6.3176,  6.3575, ... , 10.4247, 10.4618, 10.4257],
    ... ,
    [22.3772, 22.4624, 22.4227, ... , 22.5207, 22.5593, 22.5293],
    [22.5073, 22.5148, 22.5114, ... , 22.6604, 22.6344, 22.5871],
    [22.5176, 22.5275, 22.5218, ... , 22.6281, 22.6216, 22.6108]]]),
'depth': <PIL.Image.Image image mode=L size=640x480>}
```



Visual question answering (VQA)

```
from transformers import ViltProcessor, ViltForQuestionAnswering

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

question = "how many cats are there?"

encoding = processor(image, question, return_tensors="pt")
print(encoding.keys())

dict_keys(['input_ids', 'token_type_ids', 'attention_mask', 'pixel_values', 'pixel_mask'])

# forward pass
outputs = model(**encoding)
logits = outputs.logits

predicted_class_idx = logits.argmax(-1).item()
print("Predicted answer:", model.config.id2label[predicted_class_idx])
```

Predicted answer: 2

Vision Encoder-Decoder Model

- Allows to use any Transformer-based vision encoder (e.g., ViT, Swin, BEiT) with any language decoder (e.g., BERT, GPT-2)

```
from transformers import AutoTokenizer, ViTFeatureExtractor, VisionEncoderDecoderModel

repo_name = "ydshieh/vit-gpt2-coco-en"

feature_extractor = ViTFeatureExtractor.from_pretrained(repo_name)
tokenizer = AutoTokenizer.from_pretrained(repo_name)
model = VisionEncoderDecoderModel.from_pretrained(repo_name)

config.json: 100% [██████████] 4.34k/4.34k [00:00<00:00, 477kB/s]

pytorch_model.bin: 100% [██████████] 982M/982M [00:45<00:00, 21.9MB/s]

pixel_values = feature_extractor(image, return_tensors="pt").pixel_values

# autoregressively generate text (using beam search or other decoding strategy)
generated_ids = model.generate(pixel_values, max_length=16, num_beams=4, return_dict_in_generate=True)

The attention mask is not set and cannot be inferred from input because pad token is same as eos token.
We strongly recommend passing in an `attention_mask` since your input_ids may be padded. See https://hug
You may ignore this warning if your `pad_token_id` (50256) is identical to the `bos_token_id` (50256), `
```

```
# decode into text
preds = tokenizer.batch_decode(generated_ids[0], skip_special_tokens=True)
preds = [pred.strip() for pred in preds]
print(preds)

['a cat laying on top of a couch next to another cat']
```

Datasets

Main Tasks Libraries Languages Licenses Other

Modalities

3D Audio Geospatial Image
 Tabular Text Time-series Video

Size (rows)

<1K >1T

Format

json csv parquet imagefolder
 soundfolder webdataset text arrow

```
from datasets import load_dataset
dataset = load_dataset("cifar100")
```

Datasets 183,320 Filter by name

- [HuggingFaceM4/Docmatix](#)
Viewer • Updated about 19 hours ago • 1.4M • 411 • 119
- [AI-MO/NuminaMath-CoT](#)
Viewer • Updated 6 days ago • 860k • 1.38k • 87
- [5CD-AI/Viet-Doc-VQA](#)
Viewer • Updated 4 days ago • 51.9k • 12 • 51
- [AI-MO/NuminaMath-TIR](#)
Viewer • Updated 6 days ago • 72.5k • 147 • 30
- [proj-persona/PersonaHub](#)
Viewer • Updated 14 days ago • 275k • 5.11k • 240
- [mlfoundations/dclm-baseline-1.0](#)
Preview • Updated 3 days ago • 723 • 122
- [HuggingFaceTB/smollm-corpus](#)
Viewer • Updated 9 days ago • 237M • 2.54k • 114
- [Salesforce/xlam-function-calling-60k](#)
Viewer • Updated 6 days ago • 60k • 2.11k • 288
- [5CD-AI/Viet-Doc-VQA-II](#)
Viewer • Updated 4 days ago • 64.8k • 11 • 27
- [arcee-ai/The-Tome](#)
Viewer • Updated 2 days ago • 1.75M • 120 • 25

Downloading readme: 100% 9.98k/9.98k [00:00<00:00, 1.09MB/s]

Downloading data: 100% 119M/119M [00:04<00:00, 38.7MB/s]

Downloading data: 100% 23.8M/23.8M [00:01<00:00, 13.7MB/s]

Generating train split: 100% 50000/50000 [00:00<00:00, 150780.52 examples/s]

Generating test split: 100% 10000/10000 [00:00<00:00, 133679.59 examples/s]

Image Features

- Directly view images in a notebook since they are of type Image.

```
dataset['train'][0]['img']
```



```
dataset['train'].features
```

```
{'img': Image(mode=None, decode=True, id=None),
 'fine_label': ClassLabel(names=['apple', 'aquarium_fish', 'baby', 'beaver', 'bed', 'bee', 'beetle', 'bicycle', 'boat', 'camel', 'can', 'castle', 'caterpillar', 'cattle', 'chair', 'chimpanzee', 'clock', 'cloud', 'cockroach', 'couch', 'cra', 'crocodile', 'cup', 'irl', 'hamster', 'house', 'kangaroo', 'keyboard', 'lamp', 'lawn_mower', 'leopard', 'lion', 'lizard', 'lobster', 'man', 'maple', 'orange', 'orchid', 'otter', 'palm_tree', 'pear', 'pickup_truck', 'pine_tree', 'plain', 'plate', 'poppy', 'porcupine', 'pose', 'seal', 'shark', 'shrew', 'skunk', 'skyscraper', 'snail', 'snake', 'spider', 'squirrel', 'streetcar', 'sunflower', 'sweet_pepper', 'train', 'trout', 'tulip', 'turtle', 'wardrobe', 'whale', 'willow_tree', 'wolf', 'woman', 'worm'], id=None),
 'coarse_label': ClassLabel(names=['aquatic_mammals', 'fish', 'flowers', 'food_containers', 'fruit_and_vegetables', 'house_mammals', 'large_man-made_outdoor_things', 'large_natural_outdoor_scenes', 'large_omnivores_and_herbivores', 'medium_mammals', 'trees', 'vehicles_1', 'vehicles_2'], id=None)}
```

```
id2label = {id: label for id, label in enumerate(dataset['train'].features['fine_label'].names)}
print(id2label)
```

```
{0: 'apple', 1: 'aquarium_fish', 2: 'baby', 3: 'bear', 4: 'beaver', 5: 'bed', 6: 'bee', 7: 'beetle', 8: 'bicycle', 9: 'boat', 10: 'camel', 11: 'can', 12: 'castle', 13: 'caterpillar', 14: 'cattle', 15: 'chair', 16: 'chimpanzee', 17: 'clock', 18: 'cloud', 19: 'cockroach', 20: 'couch', 21: 'cra', 22: 'crocodile', 23: 'cup', 24: 'irl', 25: 'hamster', 26: 'house', 27: 'kangaroo', 28: 'keyboard', 29: 'lamp', 30: 'lawn_mower', 31: 'leopard', 32: 'lion', 33: 'lizard', 34: 'lobster', 35: 'man', 36: 'maple', 37: 'orange', 38: 'orchid', 39: 'otter', 40: 'palm_tree', 41: 'pear', 42: 'pickup_truck', 43: 'pine_tree', 44: 'plain', 45: 'plate', 46: 'poppy', 47: 'porcupine', 48: 'pose', 49: 'seal', 50: 'shark', 51: 'shrew', 52: 'skunk', 53: 'skyscraper', 54: 'snail', 55: 'snake', 56: 'spider', 57: 'squirrel', 58: 'streetcar', 59: 'sunflower', 60: 'sweet_pepper', 61: 'train', 62: 'trout', 63: 'tulip', 64: 'turtle', 65: 'wardrobe', 66: 'whale', 67: 'willow_tree', 68: 'wolf', 69: 'woman', 70: 'worm'}  
ardrobe', 95: 'whale', 96: 'willow_tree', 97: 'wolf', 98: 'woman', 99: 'worm'}
```

Image Classification

Load Food-101 dataset

```
from datasets import load_dataset  
  
food = load_dataset("food101", split="train[:10000]")  
food = food.train_test_split(test_size=0.2)
```



Preprocess

1. Load a ViT image processor

```
from transformers import AutoImageProcessor

checkpoint = "google/vit-base-patch16-224-in21k"
image_processor = AutoImageProcessor.from_pretrained(checkpoint)
```

2. Apply image transformation to images.

```
from torchvision.transforms import RandomResizedCrop, Compose, Normalize, ToTensor

normalize = Normalize(mean=image_processor.image_mean, std=image_processor.image_std)
size = ((image_processor.size["height"], image_processor.size["width"]))
_transforms = Compose([RandomResizedCrop(size), ToTensor(), normalize])
```

3. Create a preprocessing function to apply the transforms

```
def transforms(examples):
    examples["pixel_values"] = [_transforms(img.convert("RGB")) for img in examples["image"]]
    del examples["image"]
    return examples
```

Preprocess

4. To apply the preprocessing function over the entire dataset.

```
food = food.with_transform(transforms)
```

5. Create a batch of example using DefaultDataCollator

Data collators are objects that will form a batch by using a list of dataset elements as input.

```
from transformers import DefaultDataCollator  
  
data_collator = DefaultDataCollator()
```

Evaluate

1. Load a evaluate metrics

```
import evaluate

accuracy = evaluate.load("accuracy")
```

2. Create a function that passes your predictions and labels to compute.

```
import numpy as np

def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    predictions = np.argmax(predictions, axis=1)
    return accuracy.compute(predictions=predictions, references=labels)
```

Train

Auto Classes are automatically retrieve the relevant model given the name/path to the pretrained weights/config/vocabulary.

```
from transformers import AutoModelForImageClassification, TrainingArguments, Trainer

model = AutoModelForImageClassification.from_pretrained(
    checkpoint,
    num_labels=len(labels),
    id2label=id2label,
    label2id=label2id,
)
```

Train

```

training_args = TrainingArguments(
    output_dir="model",
    remove_unused_columns=False,
    eval_strategy="epoch",
    save_strategy="epoch",
    learning_rate=5e-5,
    per_device_train_batch_size=128,
    gradient_accumulation_steps=4,
    per_device_eval_batch_size=128,
    num_train_epochs=3,
    warmup_ratio=0.1,
    logging_steps=10,
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
)

trainer = Trainer(
    model=model,
    args=training_args,
    data_collator=data_collator,
    train_dataset=food["train"],
    eval_dataset=food["test"],
    tokenizer=image_processor,
    compute_metrics=compute_metrics,
)

trainer.train()

```

1. Define training hyperparameters in TrainingArguments.
2. Pass the training arguments to Trainer along with the model, dataset, tokenizer, data collator, and compute metrics function.
3. Call train()to finetune your model.

Epoch	Training Loss	Validation Loss	Accuracy
0	4.513800	4.043693	0.545000
1	3.750300	3.611866	0.776500
2	3.571900	3.495238	0.815000

Inference

```
from transformers import pipeline

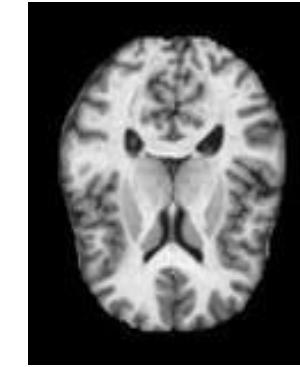
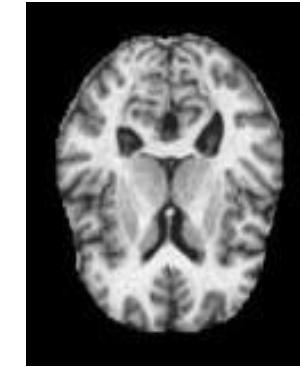
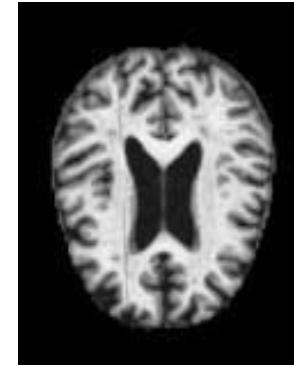
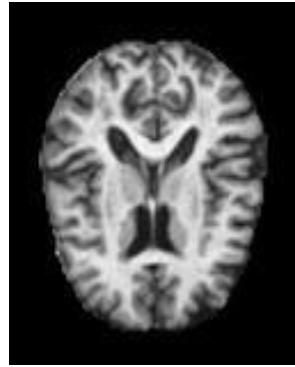
ds = load_dataset("food101", split="validation[:10]")
image = ds["image"][0]

classifier = pipeline("image-classification", model="model/checkpoint-45")
classifier(image)

[{'label': 'beignets', 'score': 0.7829036712646484},
 {'label': 'prime_rib', 'score': 0.6312826871871948},
 {'label': 'pork_chop', 'score': 0.6285311579704285},
 {'label': 'ramen', 'score': 0.6120923161506653},
 {'label': 'eggs_benedict', 'score': 0.6035012602806091}]
```



Alzheimer Disease Classification



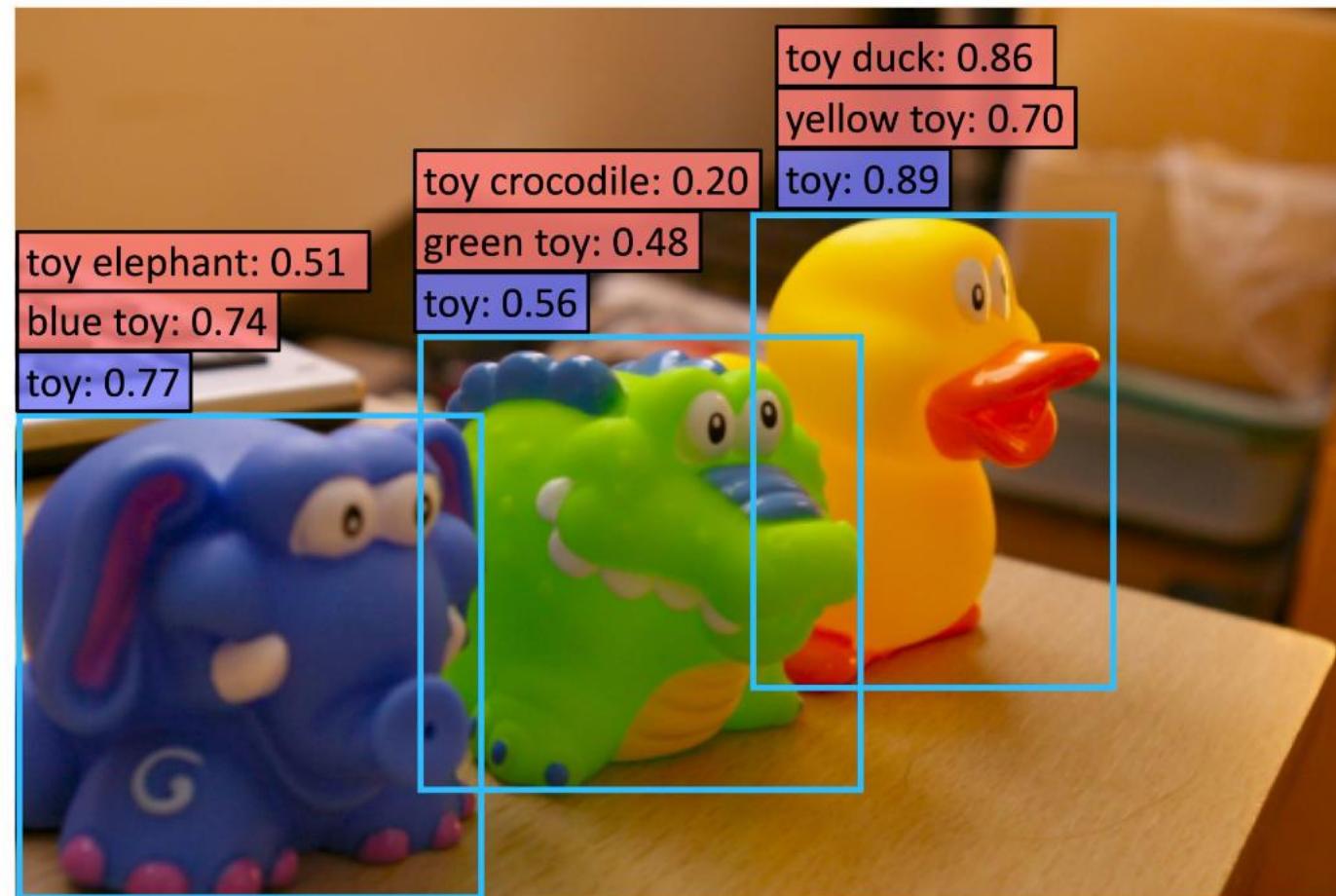
The Alzheimer_MRI Disease Classification dataset is a valuable resource for researchers and health medicine applications. This dataset focuses on the classification of Alzheimer's disease based on MRI scans. The dataset consists of brain MRI images labeled into four categories:

- '0': Mild_Demented
- '1': Moderate_Demented
- '2': Non_Demented
- '3': Very_Mild_Demented



Contrastive Language-Image Pre-training

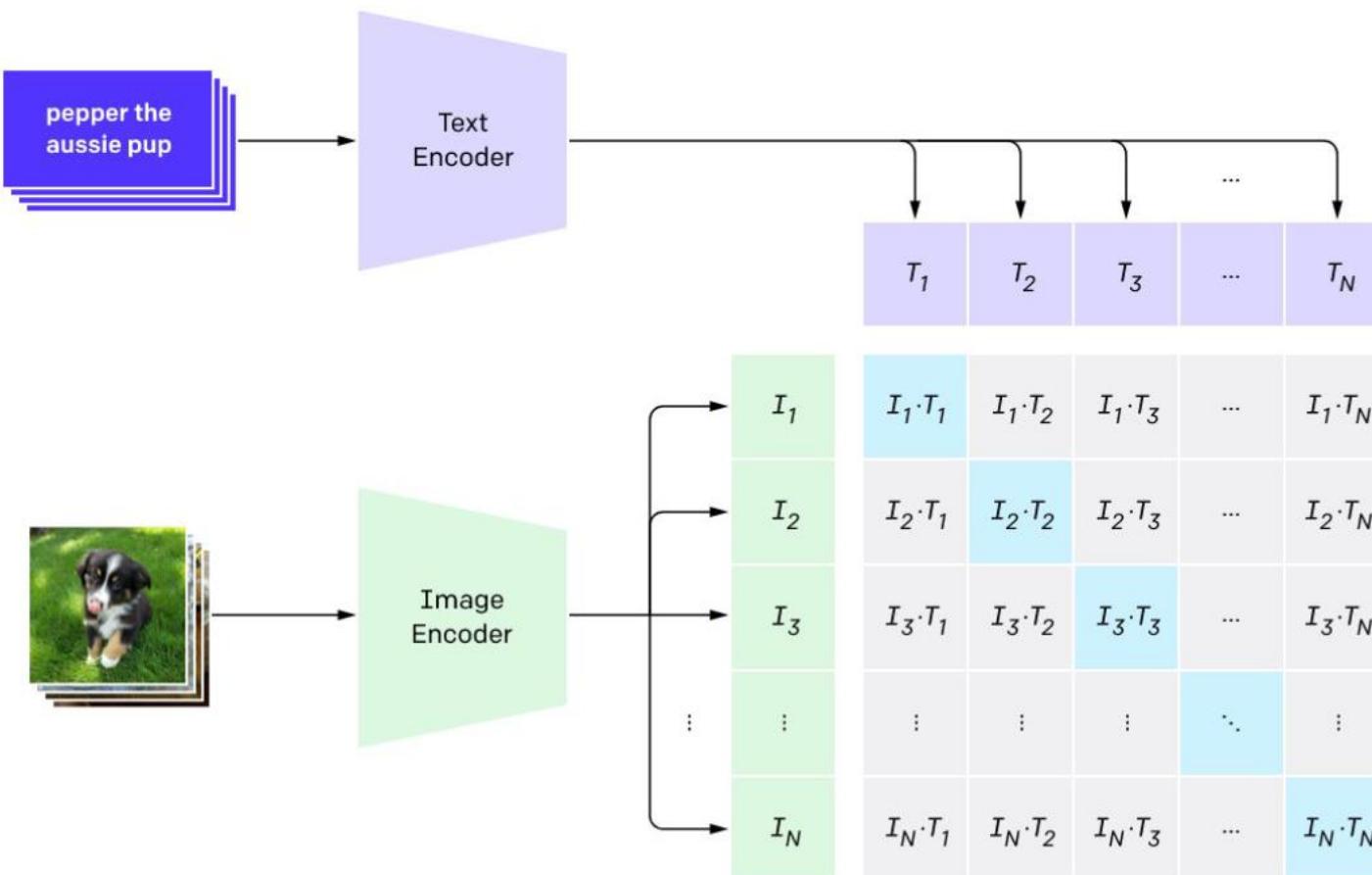
Zero-Shot Image Classification



: Novel categories : Base categories

Zero-shot classification models

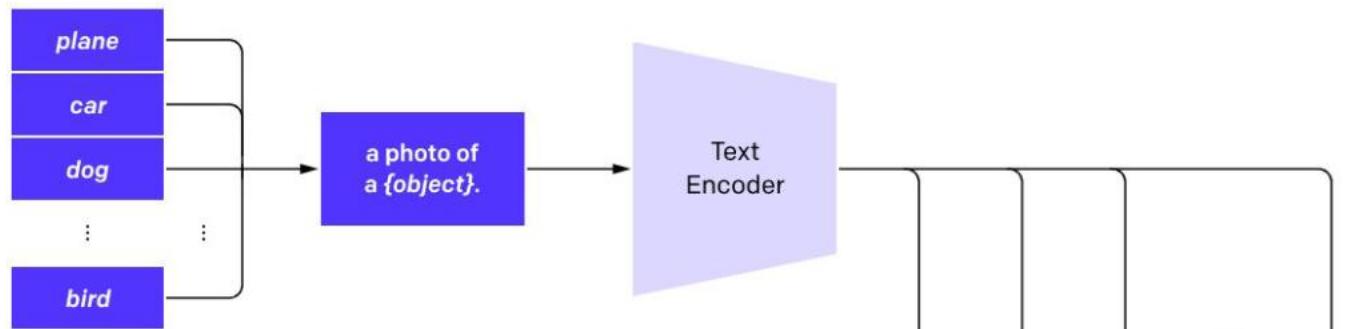
1. Contrastive pre-training



- CLIP (OpenAI)
- 400M image-text pairs

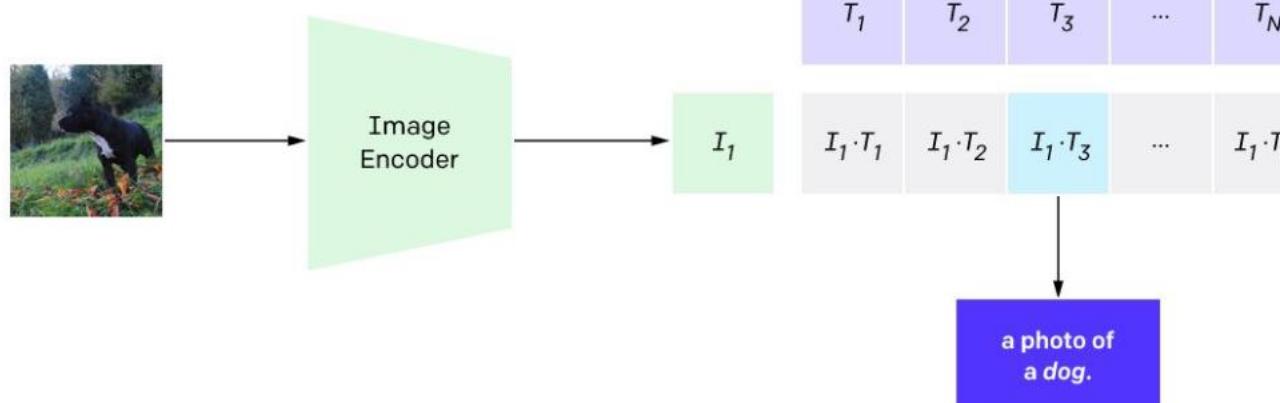
Zero-shot classification models

2. Create dataset classifier from label text



- CLIP (OpenAI)
- 76.2% Top-1 Acc on ImageNet

3. Use for zero-shot prediction



Preprocess

```
from transformers import CLIPProcessor

processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

inputs = processor(text=["a photo of a cat", "a photo of a dog"],
                    images=image,
                    return_tensors="pt",
                    padding=True)
```

```
{'input_ids': tensor([[49406, 320, 1125, 539, 320, 2368, 49407],
                     [49406, 320, 1125, 539, 320, 1929, 49407]]),
 'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1],
                           [1, 1, 1, 1, 1, 1, 1]]),
 'pixel_values': tensor([[[[ 0.5873, 0.5873, 0.6165, ..., 0.0617, 0.0471, -0.0259],
                          [ 0.5727, 0.5727, 0.6603, ..., 0.1201, 0.0763, 0.0909],
                          [ 0.5873, 0.5435, 0.6165, ..., 0.0325, 0.1201, 0.0617],
                          ...,
                          [ 1.8719, 1.8573, 1.8719, ..., 1.3902, 1.4340, 1.4194],
                          [ 1.8281, 1.8719, 1.8427, ..., 1.4486, 1.4340, 1.5070],
                          [ 1.8573, 1.9011, 1.8281, ..., 1.3756, 1.3610, 1.4486]],

                         [[-1.3169, -1.3019, -1.3169, ..., -1.4970, -1.4369, -1.4820],
                          [-1.2418, -1.2718, -1.2268, ..., -1.4369, -1.4669, -1.4519],
                          [-1.2568, -1.3169, -1.2268, ..., -1.4669, -1.4069, -1.4519],
                          ...,
                          [ 0.1239, 0.1089, 0.1239, ..., -0.7016, -0.6865, -0.6865],
                          [ 0.0789, 0.0939, 0.0488, ..., -0.6565, -0.6865, -0.6115],
                          [ 0.0939, 0.1089, 0.0038, ..., -0.7766, -0.7316, -0.6115]],

                         [[-0.4848, -0.4137, -0.3853, ..., -0.9541, -0.8545, -0.8545],
                          [-0.4137, -0.4706, -0.3711, ..., -0.8119, -0.8545, -0.7834],
                          [-0.3284, -0.4422, -0.3853, ..., -0.8688, -0.8119, -0.8830],
                          ...,
                          [ 1.5771, 1.6482, 1.6340, ..., 0.9088, 0.9514, 0.8945],
                          [ 1.6198, 1.6055, 1.6055, ..., 0.8661, 0.8092, 0.7950],
                          [ 1.6624, 1.6766, 1.5487, ..., 0.7950, 0.8661, 0.8519]]]]})}
```

Model

```
from transformers import CLIPConfig, CLIPModel

configuration = CLIPConfig()

model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")

configuration = model.config

outputs = model(**inputs)

logits_per_image = outputs.logits_per_image # this is the image-text similarity score
probs = logits_per_image.softmax(dim=1) # we can take the softmax to get the label probabilities
```

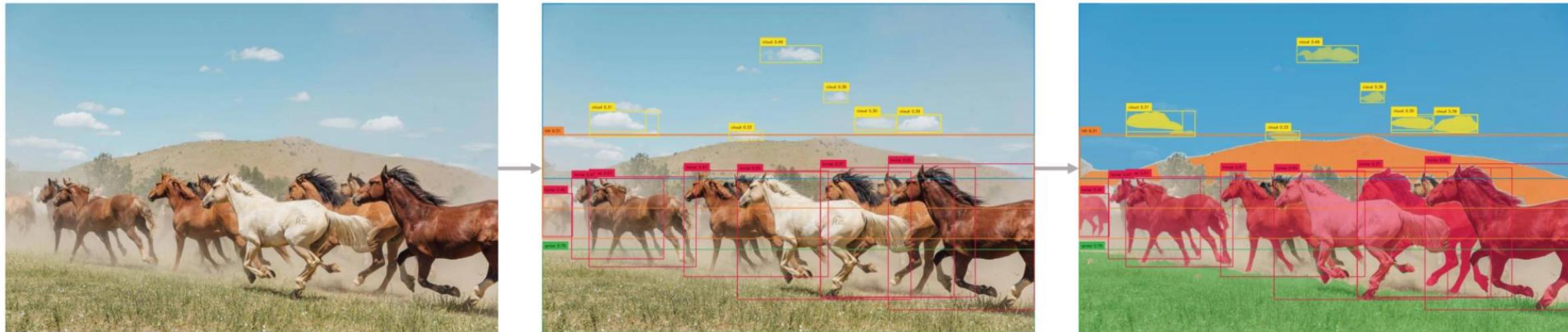
```
tensor([[24.5701, 19.3049]], grad_fn=<TBackward0>)
tensor([[0.9949, 0.0051]], grad_fn=<SoftmaxBackward0>)
```



```
inputs = processor(text=["a photo of a cat",
                      "a photo of a dog"],
                  images=image,
                  return_tensors="pt",
                  padding=True)
```

Grounding DINO with Segment Anything

Grounding DINO with Segment Anything



Text Prompt:
“Horse. Clouds. Grasses. Sky. Hill.”

Grounding DINO: Detect Everything

Grounded-SAM: Detect and Segment Everything

Zero-shot object detection

```
from transformers import pipeline

pipe = pipeline(task="zero-shot-object-detection", model="IDEA-Research/grounding-dino-base")

results = pipe('http://images.cocodataset.org/val2017/000000039769.jpg', candidate_labels=[preprocess_caption(text)], threshold=0.3)

print(results)
```



Zero-shot object detection

```
from transformers import GroundingDinoProcessor
from transformers import GroundingDinoForObjectDetection

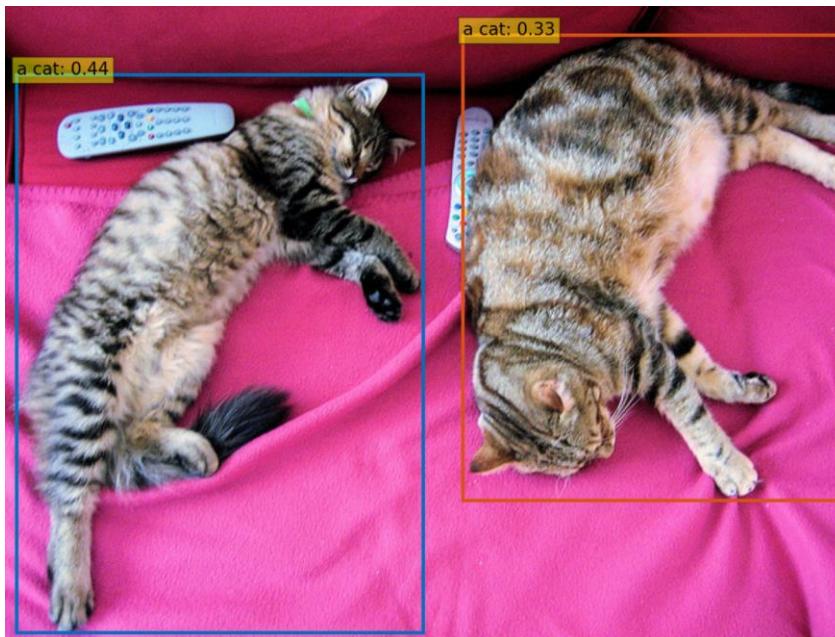
processor = GroundingDinoProcessor.from_pretrained("IDEA-Research/grounding-dino-base")
model = GroundingDinoForObjectDetection.from_pretrained("IDEA-Research/grounding-dino-base")
```

```
import torch

with torch.no_grad():
    outputs = model(**inputs)
```

Zero-shot object detection

```
# postprocess model outputs
width, height = image.size
postprocessed_outputs = processor.image_processor.post_process_object_detection(outputs,
                                target_sizes=[(height, width)],
                                threshold=0.3)
results = postprocessed_outputs[0]
```



Grounded DINO with Segment Anything

1. Use Grounding DINO to detect given set of texts in the image.
The output is a set of bounding boxes.
2. Prompt Segment Anything (SAM) with the bounding boxes, for which the model will output segmentation tasks.

Grounded DINO with Segment Anything

```
def detect(
    image: Image.Image,
    labels: List[str],
    threshold: float = 0.3,
) -> List[Dict[str, Any]]:
    """
    Use Grounding DINO to detect a set of labels in an image in a zero-shot fashion.
    """

    object_detector = pipeline(model="IDEA-Research/grounding-dino-base",
                                task="zero-shot-object-detection",
                                device='cuda')

    labels = [label if label.endswith(".") else label+"." for label in labels]

    results = object_detector(image, candidate_labels=labels, threshold=threshold)
    results = [DetectionResult.from_dict(result) for result in results]

    return results
```

1. Use Grounding DINO to detect given set of texts in the image. The output is a set of bounding boxes.

Grounded DINO with Segment Anything

```

def segment(
    image: Image.Image,
    detection_results: List[Dict[str, Any]],
) -> List[DetectionResult]:
    """
    Use Segment Anything (SAM) to generate masks given an image + a set of bounding boxes.
    """

    segmentator = AutoModelForMaskGeneration.from_pretrained('facebook/sam-vit-base').to('cuda')
    processor = AutoProcessor.from_pretrained('facebook/sam-vit-base')

    boxes = get_boxes(detection_results)
    inputs = processor(images=image, input_boxes=boxes, return_tensors="pt").to('cuda')

    outputs = segmentator(**inputs)
    masks = processor.post_process_masks(
        masks=outputs.pred_masks,
        original_sizes=inputs.original_sizes,
        reshaped_input_sizes=inputs.reshaped_input_sizes
    )[0]

    masks = refine_masks(masks)

    for detection_result, mask in zip(detection_results, masks):
        detection_result.mask = mask

    return detection_results

```

2. Prompt Segment Anything (SAM) with the bounding boxes, for which the model will output segmentation tasks.

Grounded DINO with Segment Anything

```

def grounded_segmentation(
    image: Union[Image.Image, str],
    labels: List[str],
    threshold: float = 0.3,
) -> Tuple[np.ndarray, List[DetectionResult]]:

    if isinstance(image, str):
        image = load_image(image)

    detections = detect(image, labels, threshold)
    detections = segment(image, detections)

    return np.array(image), detections

image_array, detections = grounded_segmentation(
    image=image_url,
    labels=labels,
    threshold=threshold,
    polygon_refinement=True,
    detector_id=detector_id,
    segmenter_id=segmenter_id
)

```

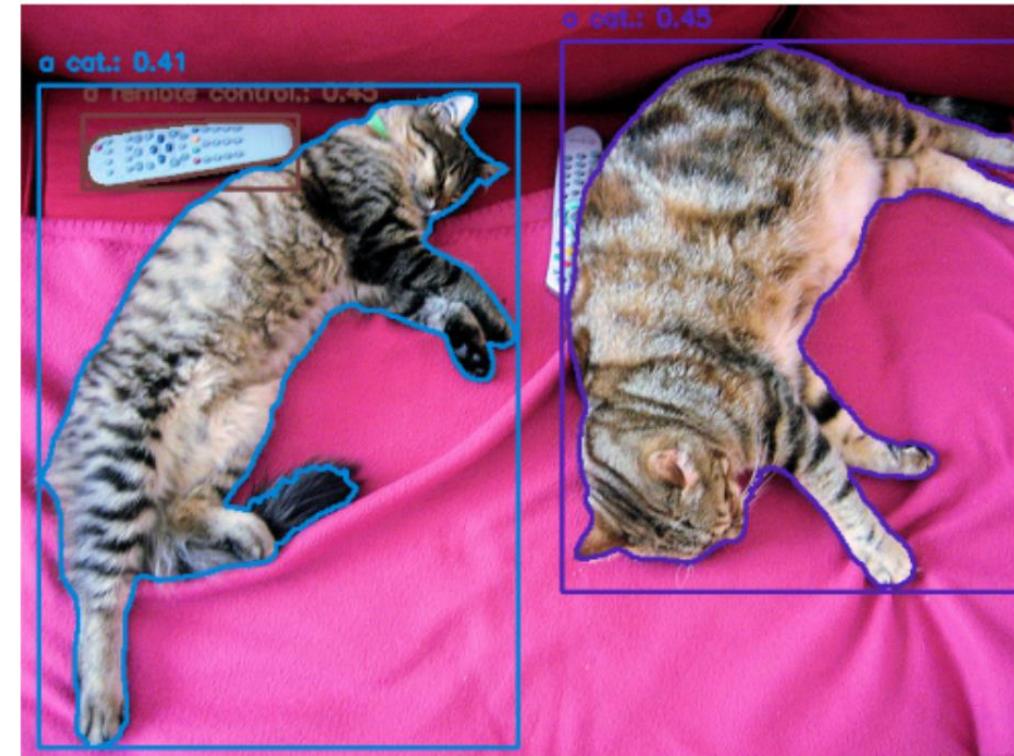
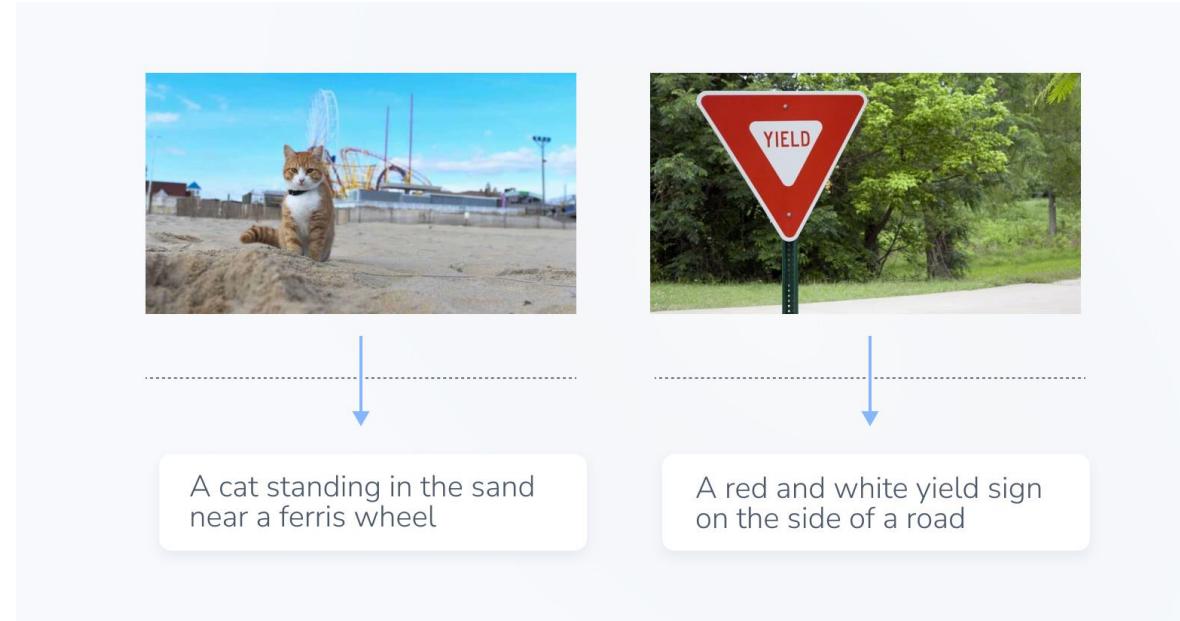


Image Captioning

Image Captioning



A cat standing in the sand
near a ferris wheel

A red and white yield sign
on the side of a road

Image captioning is the task of predicting a caption for a given image. Common real world applications of it include aiding visually impaired people that can help them navigate through different situations. Therefore, image captioning helps to improve content accessibility for people by describing images to them.

Load the data

```
from datasets import load_dataset

dataset = load_dataset("JerryMo/image-caption-blip-for-training")
dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['image', 'text'],
        num_rows: 2485
    })
})
```

```
dataset = dataset['train']
dataset[0]
```

```
{'image': <PIL.JpegImagePlugin.JpegImageFile image mode=RGB size=640x480>,
 'text': 'a family scene in the simpsons animated tv, The Simpsons'}
```



Preprocessing the data

```
from transformers import AutoProcessor

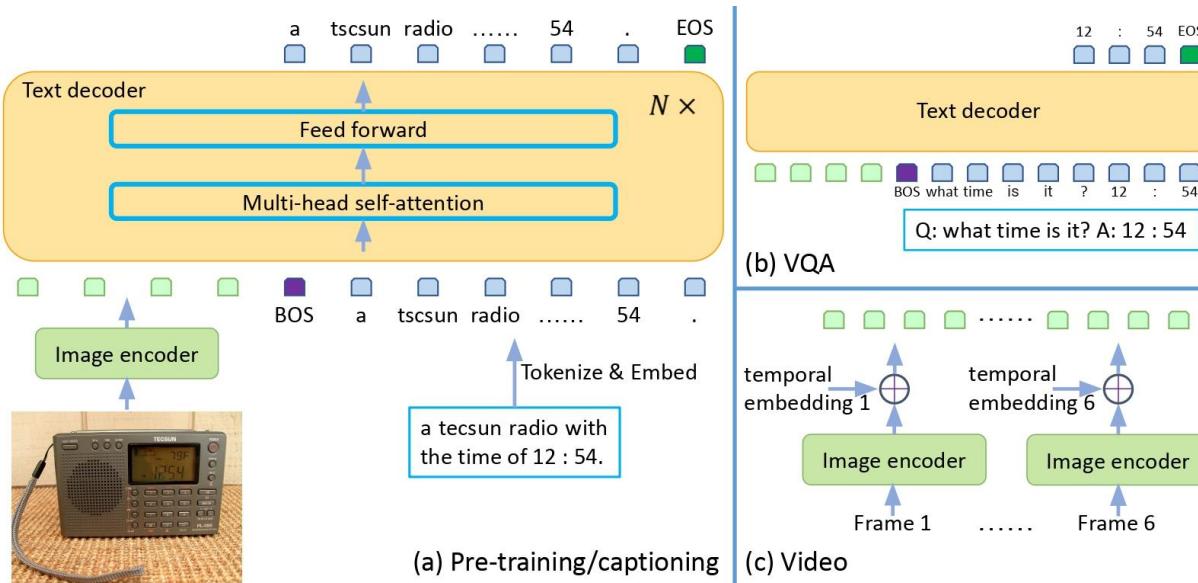
checkpoint = "microsoft/git-base"
processor = AutoProcessor.from_pretrained(checkpoint)

preprocessor_config.json: 100% [██████████] 503/503 [00:00<00:00, 56.1kB/s]
tokenizer_config.json: 100% [██████████] 453/453 [00:00<00:00, 53.1kB/s]
vocab.txt: 100% [██████████] 232k/232k [00:00<00:00, 665kB/s]
tokenizer.json: 100% [██████████] 711k/711k [00:00<00:00, 985kB/s]
special_tokens_map.json: 100% [██████████] 125/125 [00:00<00:00, 15.1kB/s]
```

```
def transforms(example_batch):
    images = [x for x in example_batch["image"]]
    captions = [x for x in example_batch["text"]]
    inputs = processor(images=images, text=captions, padding="max_length")
    inputs.update({"labels": inputs["input_ids"]})
    return inputs

train_ds.set_transform(transforms)
test_ds.set_transform(transforms)
```

Load a base model



- GIT is a Transformer decoder conditioned on both CLIP image tokens and text tokens.

```
from transformers import AutoModelForCausalLM

model = AutoModelForCausalLM.from_pretrained(checkpoint)

config.json: 100% [██████████] 2.82k/2.82k [00:00<00:00, 329kB/s]
model.safetensors: 100% [██████████] 707M/707M [00:38<00:00, 16.9MB/s]
generation_config.json: 100% [██████████] 141/141 [00:00<00:00, 16.4kB/s]
```

Evaluate

Correct text

We wanted people to know that we've got something brand new and essentially this product is uh what we call disruptive changes the way that people interact with technology.

Google output

We wanted people to know that how to me where i know and essentially this product is uh what we call scripted changes the way that people are rapid technology.

$$\text{WER} = \frac{S + D + I}{N}$$

where...

S = number of substitutions
D = number of deletions
I = number of insertions
N = number of words in the reference

```
from evaluate import load
import torch

wer = load("wer")

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predicted = logits.argmax(-1)
    decoded_labels = processor.batch_decode(labels, skip_special_tokens=True)
    decoded_predictions = processor.batch_decode(predicted, skip_special_tokens=True)
    wer_score = wer.compute(predictions=decoded_predictions, references=decoded_labels)
    return {"wer_score": wer_score}
```

Downloading builder script: 100% [4.49k/4.49k [00:00<00:00, 498kB/s]

Train

Define the training argument

```
from transformers import TrainingArguments, Trainer

model_name = checkpoint.split("/")[1]

training_args = TrainingArguments(
    output_dir=f"{model_name}-simson",
    learning_rate=5e-5,
    num_train_epochs=50,
    fp16=True,
    per_device_train_batch_size=32,
    gradient_accumulation_steps=2,
    save_total_limit=3,
    save_strategy="steps",
    save_steps=50,
    logging_steps=50,
    remove_unused_columns=False,
    label_names=["labels"],
)
```

Define the trainer and start the training

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=dataset,
    compute_metrics=compute_metrics,
)

/home/work/.conda/envs/huggingface/lib/python
se `torch.amp.GradScaler('cuda', args...)` in
    self.scaler = torch.cuda.amp.GradScaler(**k
Detected kernel version 5.4.0, which is below
version or higher.

trainer.train()
```

[11

Step	Training Loss
1	0.0000
2	0.0000
3	0.0000
4	0.0000
5	0.0000
6	0.0000
7	0.0000
8	0.0000
9	0.0000
10	0.0000
11	0.0000

Inference

```
from PIL import Image
import requests

url = "https://visla.kr/wp/wp-content/uploads/2023/11/231106_02.jpg"
image = Image.open(requests.get(url, stream=True).raw)
image
```



```
device = "cuda" if torch.cuda.is_available() else "cpu"

inputs = processor(images=image, return_tensors="pt").to(device)
pixel_values = inputs.pixel_values

generated_ids = model.generate(pixel_values=pixel_values, max_length=50)
generated_caption = processor.batch_decode(generated_ids, skip_special_tokens=True)[0]
print(generated_caption)

the simpsons character from the simpsons show, the simpsons
```

감사합니다.

Reference

<https://blog.bismart.com/en/difference-between-machine-learning-deep-learning>

<https://cs.uwaterloo.ca/~wenhuche/teaching/cs886/>

<https://huggingface.co/>

<https://www.youtube.com/watch?v=Wb5ZkZUNYc4&feature=youtu.be>