Hugging Face Transformers Tutorial

This tutorial demonstrates the key capabilities of the Hugging Face Transformers library, covering everything from simple text classification to advanced model fine-tuning and deployment. We'll explore 5 main sections:

1. **Text Classification with Pipelines** - Quick and easy sentiment analysis
2. **Direct Model Usage** - More control over tokenization and models
3. **Text Generation** - Creative AI text completion
4. **Image Segmentation** - Computer vision with semantic segmentation
5. **Model Fine-tuning & Hub Upload** - Complete ML workflow from training to sharing

Let's get started!

Section 1: Text Classification with Pipelines

The easiest way to get started with Hugging Face is using **pipelines**. Pipelines provide a high-level interface that abstracts away the complexity of tokenization, model inference, and post-processing.

In this section, we'll use a pre-trained multilingual sentiment analysis model to classify text as positive or negative. The pipeline automatically handles:

- Text tokenization
- Model inference
- Output formatting

This is perfect for quick prototyping and getting immediate results!

In [12]:
```
from rich import print
from transformers import pipeline
classifier = pipeline("text-classification",model="tabularisai/multilingual-
```

```
Device set to use mps:0
```

In [13]:
```
result=classifier("I am happy")
```

In [14]:
```
print(result)
```

```
[{'label': 'Positive', 'score': 0.43181440234184265}]
```

Section 2: Direct Model Usage

While pipelines are convenient, sometimes you need more control over the process. In this section, we'll show how to use the tokenizer and model directly.

This approach gives you:

- **Fine-grained control** over tokenization parameters
- **Access to raw model outputs** (logits, probabilities)
- **Better understanding** of what happens under the hood
- **Flexibility** to customize preprocessing and postprocessing

Let's see how to achieve the same sentiment analysis using direct model access:

In [15]:
```
from transformers import AutoTokenizer, AutoModelForSequenceClassification,
tokenizer = AutoTokenizer.from_pretrained("tabularisai/multilingual-sentimer
model = AutoModelForSequenceClassification.from_pretrained("tabularisai/mult
classifier = pipeline("text-classification", model = model, tokenizer = toke
result = classifier("I absolutely loved the new restaurant - the food was am
print(result)
```

```
Device set to use mps:0
[{'label': 'Very Positive', 'score': 0.5141021013259888}]
```

Section 3: Text Generation

Now let's explore the creative side of AI with **text generation**! This section demonstrates how to use pre-trained language models to complete text prompts.

Text generation models can:

- Complete sentences and paragraphs
- Generate creative content
- Help with writing tasks
- Provide contextual continuations

We'll use the default text generation pipeline which typically uses GPT-2 or similar models:

In [16]:
```
classifier = pipeline("text-generation")
result = classifier("I went to the mall today because")
print(result)
```

```
No model was supplied, defaulted to openai-community/gpt2 and revision 607a3
0d (https://huggingface.co/openai-community/gpt2).
Using a pipeline without specifying a model name and revision in production
is not recommended.
Device set to use mps:0
Device set to use mps:0
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```

```
[
    {
        'generated_text': "I went to the mall today because I was so e
nice friend that worked for me and she said she saw this line. It was
start your day. I'm not a huge fan of the line and my friends were sup
though, it was a great experience. I will definitely be back.\n\nI can
It is definitely worth your time and money. It's got a lot of great fo
the smell and the atmosphere. It's also a great place to get a sandwic
place to get some fresh air, and a great place to spend some time with
mall and it was the best kind of lunch I've had. It was a great option
to. The menu is a little pricey but it was worth it. The customer serv
of food. We could not wait to get a slice of the pizza and the chips.
    }
]
```

Section 4: Image Segmentation

Moving beyond text, let's explore **computer vision** with semantic segmentation! This section demonstrates how to use vision transformers for segmenting different parts of an image.

We'll use a specialized Segformer model that can identify different elements in images:

- **Skin segmentation** - Identify skin regions
- **Hair segmentation** - Detect hair areas
- **Clothing segmentation** - Recognize clothing items

This model is particularly useful for fashion, beauty, and augmented reality applications. Let's see how to segment hair from an image:

In [21]:
```
import cv2
import numpy as np
from transformers import AutoImageProcessor, SegformerForSemanticSegmentatic
import torch

# Load the processor and model
processor = AutoImageProcessor.from_pretrained("isjackwild/segformer-b0-fine
model = SegformerForSemanticSegmentation.from_pretrained("isjackwild/segform
```

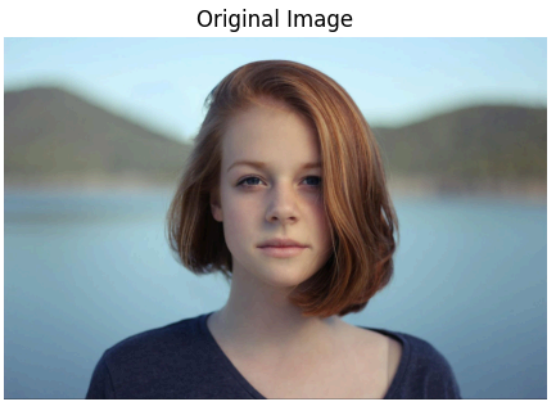
Image Processing and Visualization

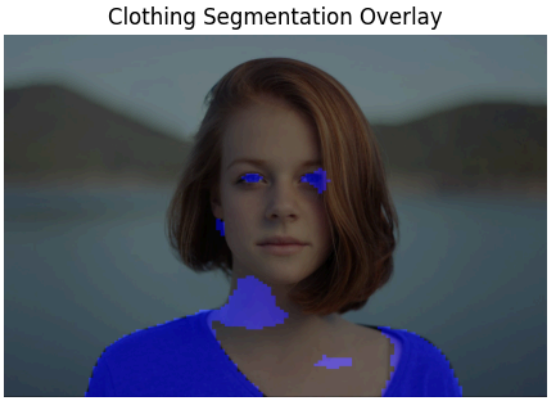
Now let's process the image through our segmentation model and visualize the results. The code below:

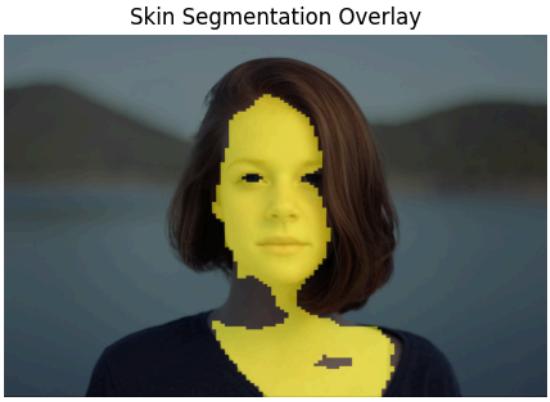
1. **Loads and preprocesses** the input image
2. **Runs inference** to get segmentation predictions
3. **Extracts regions**
4. **Creates a visual overlay** to highlight detected areas
5. **Displays results** side-by-side for comparison

In [36]:
```
import matplotlib.pyplot as plt
# Load an input image
image_path = "img.jpg" # Replace with your image path
original_image = cv2.imread(image_path)
original_image = cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB)  # Convert

# Preprocess the image
inputs = processor(images=original_image, return_tensors="pt")

# Perform inference
with torch.no_grad():
    outputs = model(**inputs)

# Get the segmentation map (logits -> argmax for predictions)
segmentation_map = torch.argmax(outputs.logits, dim=1)[0].cpu().numpy()

# Resize the segmentation map to match the original image
segmentation_map_resized = cv2.resize(segmentation_map, (original_image.shap

# Filter for hair only (class 2)
hair_mask = (segmentation_map_resized == 2).astype(np.uint8)  # Hair class i

# Create a color image for hair segmentation
hair_color = [0, 255, 0]  # Green for hair
hair_segmentation_image = np.zeros_like(original_image)
hair_segmentation_image[hair_mask == 1] = hair_color

# Blend the hair segmentation map with the original image
alpha = 0.5  # Transparency factor
overlay = cv2.addWeighted(original_image, 1 - alpha, hair_segmentation_image

clothing_mask = (segmentation_map_resized == 3).astype(np.uint8)  # Clothing
clothing_color = [0, 0, 255]  # Blue for clothing
clothing_segmentation_image = np.zeros_like(original_image)
clothing_segmentation_image[clothing_mask == 1] = clothing_color
clothing_segmentation_overlay = cv2.addWeighted(original_image, 1 - alpha, c

skin_mask = (segmentation_map_resized == 1).astype(np.uint8)  # Skin class i
skin_color = [255, 255, 0]  # Yellow for skin
skin_segmentation_image = np.zeros_like(original_image)
skin_segmentation_image[skin_mask == 1] = skin_color
skin_segmentation_overlay = cv2.addWeighted(original_image, 1 - alpha, skin_

images = [
    (original_image, "Original Image"),
    (overlay, "Hair Segmentation Overlay"),
    (clothing_segmentation_overlay, "Clothing Segmentation Overlay"),
    (skin_segmentation_overlay, "Skin Segmentation Overlay")
]

fig, axes = plt.subplots(2, 2, figsize=(12, 12))

for ax, (img, title) in zip(axes.ravel(), images):
    ax.imshow(img)
```

```
    ax.set_title(title)
    ax.axis("off")
```







Section 5: Model Fine-tuning & Hub Upload

In this final section, we'll demonstrate the complete machine learning workflow: from loading a dataset to fine-tuning a model and sharing it on the Hugging Face Hub.

This section covers:

- **Authentication** with Hugging Face Hub
- **Dataset loading** and preprocessing
- **Model fine-tuning** with the Trainer API
- **Publishing models** to share with the community

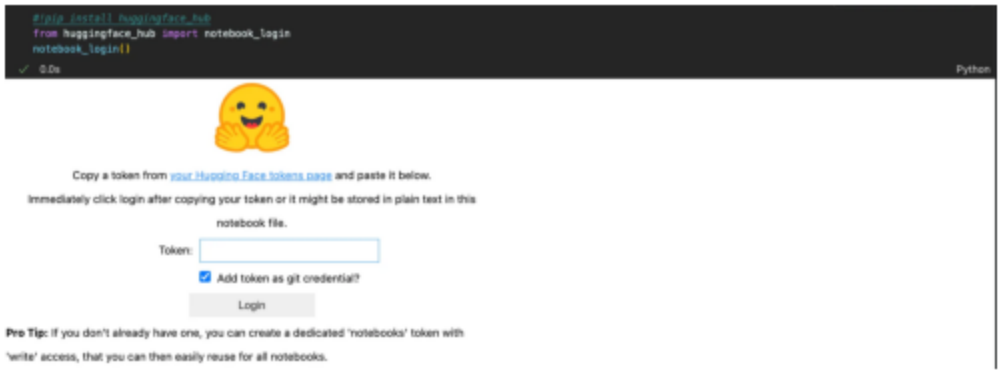
Let's fine-tune a small BERT model on sentiment analysis and upload it to the Hub:

Authentication with Hugging Face Hub

In [ ]:
```
from huggingface_hub import notebook_login
notebook_login()
```

```
VBox(children=(HTML(value='<center> <img\nsrc=https://huggingface.co/front/assets/huggingface_logo-noborder.sv…
```

In [41]:
```
img_bgr = cv2.imread("popup.png")
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
plt.imshow(img_rgb)
plt.axis("off")
plt.show()
```



Dataset loading and preprocessing

In [6]:
```
from datasets import load_dataset
from transformers import AutoTokenizer, AutoModelForSequenceClassification

# Load a small dataset (e.g., Stanford Sentiment Treebank)
dataset = load_dataset("glue", "sst2")

# Define the model checkpoint
model_name = "prajjwal1/bert-tiny"

# Load the tokenizer and the model
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_l
```

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at prajjwal1/bert-tiny and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

In [9]:
```
# Create a tokenization function
def tokenize_function(examples):
    return tokenizer(examples["sentence"], truncation=True,padding="max_leng

# Apply the tokenizer to the entire dataset
tokenized_datasets = dataset.map(tokenize_function, batched=True)

# For a quick demo, create small training and validation subsets
```

```
small_train_dataset = tokenized_datasets["train"].shuffle(seed=42).select(ra
small_eval_dataset = tokenized_datasets["validation"].shuffle(seed=42).selec
```

```
Map:   0%|          | 0/67349 [00:00<?, ? examples/s]
Map:   0%|          | 0/872 [00:00<?, ? examples/s]
Map:   0%|          | 0/1821 [00:00<?, ? examples/s]
```

Model fine-tuning with the Trainer API

In [10]:

```
from transformers import TrainingArguments, Trainer

# Define a name for your model repository on the Hub
model_repo_name = "bert-tiny-finetuned-sst2-demo"

# Set the training arguments
training_args = TrainingArguments(
    output_dir=model_repo_name,
    num_train_epochs=1,              # 1 epoch is enough for a quick demo
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    eval_strategy="epoch",     # Evaluate at the end of each epoch
    logging_dir='./logs',
    push_to_hub=True,                # This will upload the model to the Hub
)

# Create the Trainer instance
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=small_train_dataset,
    eval_dataset=small_eval_dataset,
)

# Start fine-tuning!
trainer.train()
```

```
/Users/lokeshmanideep/Downloads/Masters/FallSemester2025/CS421/HuggingfaceTu
torial/env/lib/python3.11/site-packages/torch/utils/data/dataloader.py:684:
UserWarning: 'pin_memory' argument is set as true but not supported on MPS n
ow, then device pinned memory won't be used.
  warnings.warn(warn_msg)
```


[63/63 00:01, Epoch 1/1]

Epoch	Training Loss	Validation Loss
1	No log	0.685798

```
/Users/lokeshmanideep/Downloads/Masters/FallSemester2025/CS421/HuggingfaceTu
torial/env/lib/python3.11/site-packages/torch/utils/data/dataloader.py:684:
UserWarning: 'pin_memory' argument is set as true but not supported on MPS n
ow, then device pinned memory won't be used.
  warnings.warn(warn_msg)
```

Out[10]:
```
TrainOutput(global_step=63, training_loss=0.6856122092595176, metrics={'tra
in_runtime': 6.1848, 'train_samples_per_second': 161.686, 'train_steps_per_
second': 10.186, 'total_flos': 317621760000.0, 'train_loss': 0.685612209259
5176, 'epoch': 1.0})
```

Publishing models to share with the community

In [11]:
```
trainer.push_to_hub(commit_message="End of training")
```

```
Processing Files (0 / 0): |          |  0.00B /  0.00B
New Data Upload: |          |  0.00B /  0.00B
```

Out[11]:
```
CommitInfo(commit_url='https://huggingface.co/lbogg/bert-tiny-finetuned-sst
2-demo/commit/bb45a1201e62c5910453ae1b102e49e75250dffa', commit_message='En
d of training', commit_description='', oid='bb45a1201e62c5910453ae1b102e49e
75250dffa', pr_url=None, repo_url=RepoUrl('https://huggingface.co/lbogg/ber
t-tiny-finetuned-sst2-demo', endpoint='https://huggingface.co', repo_type
='model', repo_id='lbogg/bert-tiny-finetuned-sst2-demo'), pr_revision=None,
pr_num=None)
```