```
!pip install -q transformers torch gradio
```

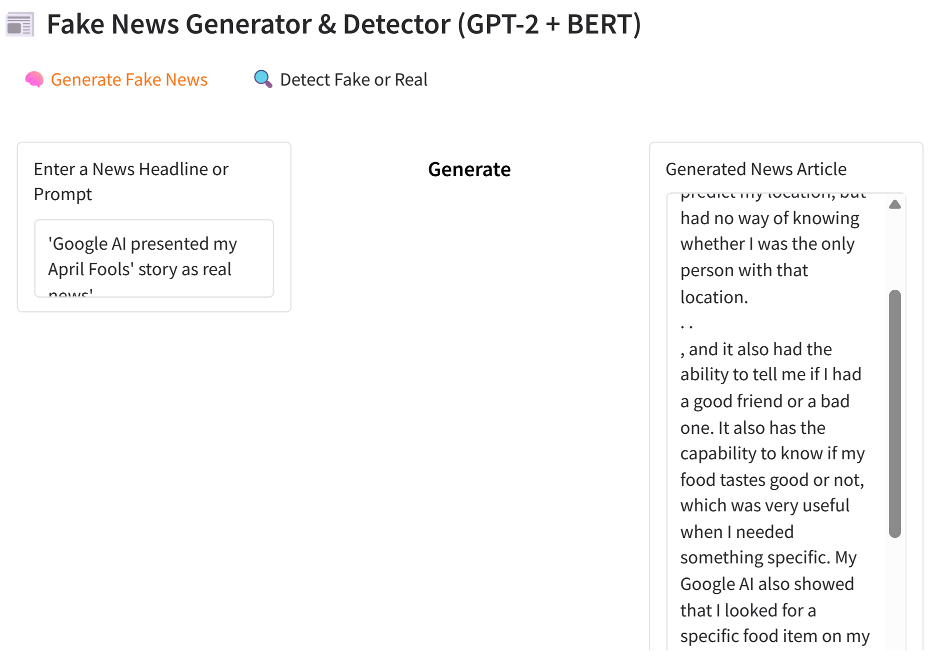
```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 3.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 42.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 19.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 19.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 2.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 5.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 11.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 9.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 5.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 42.9 MB/s eta 0:00:00
```

```
import torch
from transformers import GPT2LMHeadModel, GPT2Tokenizer
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import gradio as gr

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


gpt2_tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
gpt2_model = GPT2LMHeadModel.from_pretrained("gpt2").to(device)

bert_tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
bert_model = AutoModelForSequenceClassification.from_pretrained("bert-base-uncased", num_labels = 2).to(device)
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as :
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json:  100%                    26.0/26.0  [00:00<00:00,  1.89kB/s]
vocab.json:  100%                               1.04M/1.04M  [00:00<00:00,  13.5MB/s]
merges.txt:  100%                               456k/456k  [00:00<00:00,  17.9MB/s]
tokenizer.json:  100%                           1.36M/1.36M  [00:00<00:00,  26.1MB/s]
config.json:  100%                              665/665  [00:00<00:00,  56.7kB/s]
model.safetensors:  100%                        548M/548M  [00:13<00:00,  45.7MB/s]
generation_config.json:  100%                   124/124  [00:00<00:00,  2.18kB/s]
tokenizer_config.json:  100%                    48.0/48.0  [00:00<00:00,  1.11kB/s]
config.json:  100%                              570/570  [00:00<00:00,  9.60kB/s]
vocab.txt:  100%                                232k/232k  [00:00<00:00,  3.92MB/s]
tokenizer.json:  100%                           466k/466k  [00:00<00:00,  6.81MB/s]
model.safetensors:  100%                        440M/440M  [00:11<00:00,  56.9MB/s]
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly ini
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
def generate_fake_news(prompt):
    inputs = gpt2_tokenizer.encode(prompt, return_tensors="pt").to(device)
    outputs = gpt2_model.generate(
        inputs,
        max_length=200,
        num_return_sequences=1,
        no_repeat_ngram_size=2,
        do_sample=True,
        temperature=0.7,
        top_k=50,
        top_p=0.95,
        early_stopping=True
    )
    generated_text = gpt2_tokenizer.decode(outputs[0], skip_special_tokens=True)
    return generated_text


def detect_news(text):
    inputs = bert_tokenizer(text, return_tensors="pt", truncation=True, padding=True).to(device)
    with torch.no_grad():
        outputs = bert_model(**inputs)
        logits = outputs.logits
        predicted_class = torch.argmax(logits, dim=1).item()
        confidence = torch.softmax(logits, dim=1)[0][predicted_class].item()
        label = "Fake News" if predicted_class == 0 else "Real News"
    return f"{label} (Confidence: {confidence:.2f})"


with gr.Blocks() as demo:
    gr.Markdown("## 📰 Fake News Generator & Detector (GPT-2 + BERT)")

    with gr.Tab("🧠 Generate Fake News"):
        with gr.Row():
            input_text = gr.Textbox(label="Enter a News Headline or Prompt", placeholder="e.g. A mysterious object was spotted in the sk
            generate_btn = gr.Button("Generate")
            output_text = gr.Textbox(label="Generated News Article")
            generate_btn.click(generate_fake_news, inputs=input_text, outputs=output_text)

    with gr.Tab("🔍 Detect Fake or Real"):
        with gr.Row():
            detect_input = gr.Textbox(label="Enter a News Article or Statement", placeholder="Paste a paragraph to detect if it's fake c
            detect_btn = gr.Button("Detect")
            detect_output = gr.Textbox(label="Detection Result")
            detect_btn.click(detect_news, inputs=detect_input, outputs=detect_output)

demo.launch()
```

```
It looks like you are running Gradio on a hosted Jupyter notebook, which requires `share=True`. Automatically setting `share=True`

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://908cde1e4326ee6673.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working
```



📰 Fake News Generator & Detector (GPT-2 + BERT)

🧠 Generate Fake News   🔍 Detect Fake or Real

Enter a News Headline or Prompt

'Google AI presented my April Fools' story as real news'

Generate

Generated News Article

predict my location, but had no way of knowing whether I was the only person with that location.
. .
, and it also had the ability to tell me if I had a good friend or a bad one. It also has the capability to know if my food tastes good or not, which was very useful when I needed something specific. My Google AI also showed that I looked for a specific food item on my

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```