```
import torch
from transformers import BartTokenizer, BartForConditionalGeneration


# 1. Load tokenizer and model
tokenizer = BartTokenizer.from_pretrained("facebook/bart-base")
model = BartForConditionalGeneration.from_pretrained("facebook/bart-base")
model.eval()
```

```
/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),
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(
vocab.json:        899k/? [00:00<00:00, 3.15MB/s]
merges.txt:        456k/? [00:00<00:00, 7.81MB/s]
tokenizer.json:        1.36M/? [00:00<00:00, 9.81MB/s]
config.json:        1.72k/? [00:00<00:00, 23.1kB/s]
model.safetensors: 100%                                                558M/558M [00:21<00:00, 24.5MB/s]
BartForConditionalGeneration(
  (model): BartModel(
    (shared): BartScaledWordEmbedding(50265, 768, padding_idx=1)
    (encoder): BartEncoder(
      (embed_tokens): BartScaledWordEmbedding(50265, 768, padding_idx=1)
      (embed_positions): BartLearnedPositionalEmbedding(1026, 768)
      (layers): ModuleList(
        (0-5): 6 x BartEncoderLayer(
          (self_attn): BartAttention(
            (k_proj): Linear(in_features=768, out_features=768, bias=True)
            (v_proj): Linear(in_features=768, out_features=768, bias=True)
            (q_proj): Linear(in_features=768, out_features=768, bias=True)
            (out_proj): Linear(in_features=768, out_features=768, bias=True)
          )
          (self_attn_layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
          (activation_fn): GELUActivation()
          (fc1): Linear(in_features=768, out_features=3072, bias=True)
          (fc2): Linear(in_features=3072, out_features=768, bias=True)
          (final_layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        )
      )
      (layernorm_embedding): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    )
    (decoder): BartDecoder(
      (embed_tokens): BartScaledWordEmbedding(50265, 768, padding_idx=1)
      (embed_positions): BartLearnedPositionalEmbedding(1026, 768)
      (layers): ModuleList(
        (0-5): 6 x BartDecoderLayer(
          (self_attn): BartAttention(
            (k_proj): Linear(in_features=768, out_features=768, bias=True)
            (v_proj): Linear(in_features=768, out_features=768, bias=True)
            (q_proj): Linear(in_features=768, out_features=768, bias=True)
            (out_proj): Linear(in_features=768, out_features=768, bias=True)
          )
          (activation_fn): GELUActivation()
          (self_attn_layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
          (encoder_attn): BartAttention(
            (k_proj): Linear(in_features=768, out_features=768, bias=True)
            (v_proj): Linear(in_features=768, out_features=768, bias=True)
            (q_proj): Linear(in_features=768, out_features=768, bias=True)
            (out_proj): Linear(in_features=768, out_features=768, bias=True)
          )
          (encoder_attn_layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
          (fc1): Linear(in_features=768, out_features=3072, bias=True)
          (fc2): Linear(in_features=3072, out_features=768, bias=True)
          (final_layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        )
      )
      (layernorm_embedding): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    )
  )
  (lm_head): Linear(in_features=768, out_features=50265, bias=False)
)
```

```
# 2. Input text
text = "The quick brown fox jumps over the lazy dog."
inputs = tokenizer(text, retu
```



```
# 3. Get encoder hidden states (text → latent)
with torch.no_grad():
```

```
    encoder_outputs = model.model.encoder(
        input_ids=inputs["input_ids"],
        attention_mask=inputs["attention_mask"]
    )
    latent = encoder_outputs.last_hidden_state  # shape: [1, seq_len, hidden_dim]


predicted_latent = latent.clone()


# 4. Begin decoding autoregressively from <s> token (BOS)
decoder_input_ids = torch.tensor([[tokenizer.bos_token_id]])

# Generate text token-by-token
generated_tokens = []
max_len = 32  # max tokens to generate

with torch.no_grad():
    for _ in range(max_len):
        outputs = model(
            encoder_outputs=(predicted_latent,),
            decoder_input_ids=decoder_input_ids
        )
        next_token_logits = outputs.logits[:, -1, :]
        next_token = torch.argmax(next_token_logits, dim=-1)

        if next_token.item() == tokenizer.eos_token_id:
            break

        generated_tokens.append(next_token.item())
        decoder_input_ids = torch.cat([decoder_input_ids, next_token.unsqueeze(0)], dim=1)

len(generated_tokens)
```

```
11
```

```
# Decode output tokens to text
output_text = tokenizer.decode(generated_tokens, skip_special_tokens=True)


print("Original Text:")
print(text)

print()

print("Generated Text:")
print(output_text)
```

```
Original Text:
The quick brown fox jumps over the lazy dog.

Generated Text:
The quick brown fox jumps over the lazy dog.
```