In [1]:

```python
from transformers import BertTokenizer, BertForMaskedLM
import torch

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForMaskedLM.from_pretrained('bert-base-uncased')
model.eval()
```

```
model.safetensors:   0%|          | 0.00/440M [00:00<?, ?B/s]
```

```
C:\Users\ravit\AppData\Local\Programs\Python\Python311\Lib\site-packages\huggingface_hub\file_download.py:157: UserWarning: `huggingface_hub` cache-system uses symlinks by default to efficiently store duplicated files but your machine does not support them in C:\Users\ravit\.cache\huggingface\hub\models--bert-base-uncased. Caching files will still work but in a degraded version that might require more space on your disk. This warning can be disabled by setting the `HF_HUB_DISABLE_SYMLINKS_WARNING` environment variable. For more details, see https://huggingface.co/docs/huggingface_hub/how-to-cache#limitations.
To support symlinks on Windows, you either need to activate Developer Mode or to run Python as an administrator. In order to see activate developer mode, see this article: https://docs.microsoft.com/en-us/windows/apps/get-started/enable-your-device-for-development
  warnings.warn(message)
Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertForMaskedLM: ['bert.pooler.dense.bias', 'bert.pooler.dense.weight', 'cls.seq_relationship.bias', 'cls.seq_relationship.weight']
- This IS expected if you are initializing BertForMaskedLM from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForMaskedLM from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
```

Out[1]:

```
BertForMaskedLM(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSdpaSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
  )
  (cls): BertOnlyMLMHead(
    (predictions): BertLMPredictionHead(
      (transform): BertPredictionHeadTransform(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (transform_act_fn): GELUActivation()
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      )
      (decoder): Linear(in_features=768, out_features=30522, bias=True)
    )
  )
)
```

In [2]:

```python
text = "In Paris, I love to visit the [MASK] Tower and enjoy [MASK] cuisine."
#encode() function used to generate a token id's
#When you set return_tensors='pt', the encode function will return a dictionary with the following keys:
#input_ids: A PyTorch tensor containing the encoded input IDs.
#attention_mask: A PyTorch tensor containing the attention mask.
input_ids = tokenizer.encode(text, return_tensors='pt')
#In PyTorch, the torch.no_grad() function is used to disable gradient computation for a given code block.
#This can be useful in several scenarios, such as when you want to perform inference on a model without updating its weights.
#no_grad() function:-It will reduce memory consumption for computations that would otherwise have requires_grad=True.
#torch.no_grad() disables computing gradients
with torch.no_grad():
    outputs = model(input_ids)
    predictions = outputs[0]
results = []
for i, mask_id in enumerate(torch.where(input_ids == tokenizer.mask_token_id)[1]):
    predicted_index = torch.argmax(predictions[0, mask_id]).item()
    predicted_word = tokenizer.decode([predicted_index])
    print(f"Prediction for MASK {i+1}: {predicted_word}")
    results.append(predicted_word)

print("Initial Text:", text)
predicted_text = text
for result in results:
    predicted_text = predicted_text.replace("[MASK]", result, 1)
print("Predicted Text", predicted_text)
#torch.argmax(next_token_logits, dim=-1)” means that we’re selecting the token with highest probability.
#Then, in the next line, we take this chosen token and add it to our input sentence to keep generating text.
#So, we’re building our text step by step, always choosing the word that seems the most probable according to the model.
#The items () method in the dictionary is used to return each item in a dictionary as tuples in a list
```

```
Prediction for MASK 1: clock
Prediction for MASK 2: its
Initial Text: In Paris, I love to visit the [MASK] Tower and enjoy [MASK] cuisine.
Predicted Text In Paris, I love to visit the clock Tower and enjoy its cuisine.
```

In [ ]:

In [ ]: