1. Cài đặt thư viện

```
# Cài đặt thư viện transformers và torch
!pip install transformers torch
```

```
Requirement already satisfied: transformers in
/usr/local/lib/python3.12/dist-packages (4.57.1)
Requirement already satisfied: torch in
/usr/local/lib/python3.12/dist-packages (2.8.0+cu126)
Requirement already satisfied: filelock in
/usr/local/lib/python3.12/dist-packages (from transformers) (3.20.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.34.0 in
/usr/local/lib/python3.12/dist-packages (from transformers) (0.36.0)
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.12/dist-packages (from transformers) (2.0.2)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.12/dist-packages (from transformers) (25.0)
Requirement already satisfied: pyyaml>=5.1 in
/usr/local/lib/python3.12/dist-packages (from transformers) (6.0.3)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.12/dist-packages (from transformers)
(2024.11.6)
Requirement already satisfied: requests in
/usr/local/lib/python3.12/dist-packages (from transformers) (2.32.4)
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in
/usr/local/lib/python3.12/dist-packages (from transformers) (0.22.1)
Requirement already satisfied: safetensors>=0.4.3 in
/usr/local/lib/python3.12/dist-packages (from transformers) (0.6.2)
Requirement already satisfied: tqdm>=4.27 in
/usr/local/lib/python3.12/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: typing-extensions>=4.10.0 in
/usr/local/lib/python3.12/dist-packages (from torch) (4.15.0)
Requirement already satisfied: setuptools in
/usr/local/lib/python3.12/dist-packages (from torch) (75.2.0)
Requirement already satisfied: sympy>=1.13.3 in
/usr/local/lib/python3.12/dist-packages (from torch) (1.13.3)
Requirement already satisfied: networkx in
/usr/local/lib/python3.12/dist-packages (from torch) (3.5)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.12/dist-packages (from torch) (3.1.6)
Requirement already satisfied: fsspec in
/usr/local/lib/python3.12/dist-packages (from torch) (2025.3.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.80)
```

```
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in
/usr/local/lib/python3.12/dist-packages (from torch) (9.10.2.21)
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.4.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in
/usr/local/lib/python3.12/dist-packages (from torch) (11.3.0.4)
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in
/usr/local/lib/python3.12/dist-packages (from torch) (10.3.7.77)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in
/usr/local/lib/python3.12/dist-packages (from torch) (11.7.1.2)
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.5.4.2)
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in
/usr/local/lib/python3.12/dist-packages (from torch) (0.7.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in
/usr/local/lib/python3.12/dist-packages (from torch) (2.27.3)
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in
/usr/local/lib/python3.12/dist-packages (from torch) (12.6.85)
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in
/usr/local/lib/python3.12/dist-packages (from torch) (1.11.1.6)
Requirement already satisfied: triton==3.4.0 in
/usr/local/lib/python3.12/dist-packages (from torch) (3.4.0)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in
/usr/local/lib/python3.12/dist-packages (from huggingface-
hub<1.0,>=0.34.0->transformers) (1.2.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3->torch)
(1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.12/dist-packages (from jinja2->torch) (3.0.3)
Requirement already satisfied: charset_normalizer<4,>=2 in
/usr/local/lib/python3.12/dist-packages (from requests->transformers)
(3.4.4)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.12/dist-packages (from requests->transformers)
(3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.12/dist-packages (from requests->transformers)
(2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.12/dist-packages (from requests->transformers)
(2025.10.5)
```

2. Bài 1: Khôi phục Masked Token (Masked Language Modeling)

Sử dụng mô hình BERT (Encoder-only) để điền từ vào chỗ trống.

1. Tải pipeline "fill-mask"

```
from transformers import pipeline

# Mặc định pipeline này sẽ tải mô hình distilroberta-base (hoặc biến thể BERT tương tự)
mask_filler = pipeline("fill-mask")
```

```
No model was supplied, defaulted to distilbert/distilroberta-base and revision fb53ab8 (https://huggingface.co/distilbert/distilroberta-base).
Using a pipeline without specifying a model name and revision in production is not recommended.
Some weights of the model checkpoint at distilbert/distilroberta-base were not used when initializing RobertaForMaskedLM: ['roberta.pooler.dense.bias', 'roberta.pooler.dense.weight']
- This IS expected if you are initializing RobertaForMaskedLM 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 RobertaForMaskedLM from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
Device set to use cpu
```

2. Câu đầu vào với token [MASK]

```
# Lưu ý: Một số model dùng <mask> thay vì [MASK], nhưng pipeline thường tự xử lý
input_sentence = "Hanoi is the <mask> of Vietnam."
# Nếu code báo lỗi, hãy thử thay <mask> bằng [MASK] tùy theo model mặc định được tải về

input_sentence
```

```
{"type":"string"}
```

3. Thực hiện dự đoán (top_k=5)

```
predictions = mask_filler(input_sentence, top_k=5)

predictions
```

```
[{'score': 0.9341353178024292,
  'token': 812,
  'token_str': ' capital',
  'sequence': 'Hanoi is the capital of Vietnam.'},
 {'score': 0.02996085211634636,
  'token': 3497,
  'token_str': ' Republic',
  'sequence': 'Hanoi is the Republic of Vietnam.'},
 {'score': 0.010538393631577492,
  'token': 1867,
  'token_str': ' Capital',
  'sequence': 'Hanoi is the Capital of Vietnam.'},
 {'score': 0.005418903660029173,
  'token': 32357,
  'token_str': ' birthplace',
  'sequence': 'Hanoi is the birthplace of Vietnam.'},
 {'score': 0.0014444863190874457,
  'token': 1144,
  'token_str': ' heart',
  'sequence': 'Hanoi is the heart of Vietnam.'}]
```

4. In kết quả

```
print(f"Câu gốc: {input_sentence}")
print("-" * 30)
for pred in predictions:
    print(f"Từ dự đoán: '{pred['token_str']}' | Độ tin cậy:
{pred['score']:.4f}")
    print(f" -> Câu hoàn chỉnh: {pred['sequence']}")
```

```
Câu gốc: Hanoi is the <mask> of Vietnam.
------------------------------
Từ dự đoán: ' capital' | Độ tin cậy: 0.9341
 -> Câu hoàn chỉnh: Hanoi is the capital of Vietnam.
Từ dự đoán: ' Republic' | Độ tin cậy: 0.0300
 -> Câu hoàn chỉnh: Hanoi is the Republic of Vietnam.
Từ dự đoán: ' Capital' | Độ tin cậy: 0.0105
 -> Câu hoàn chỉnh: Hanoi is the Capital of Vietnam.
Từ dự đoán: ' birthplace' | Độ tin cậy: 0.0054
 -> Câu hoàn chỉnh: Hanoi is the birthplace of Vietnam.
Từ dự đoán: ' heart' | Độ tin cậy: 0.0014
 -> Câu hoàn chỉnh: Hanoi is the heart of Vietnam.
```

Trả lời câu hỏi Bài 1:

Mô hình có dự đoán đúng từ "capital" không?

=> Trả lời: Có (thường "capital" sẽ là dự đoán có độ tin cậy cao nhất).

Tại sao các mô hình Encoder-only như BERT lại phù hợp cho tác vụ này?

=> Trả lời: Vì kiến trúc Encoder có khả năng nhìn hai chiều (bidirectional). Để điền vào chỗ trống ở giữa câu ("Hanoi is the ... of Vietnam"), mô hình cần hiểu ngữ cảnh từ cả bên trái ("Hanoi") và bên phải ("Vietnam"). Encoder được thiết kế để làm chính xác việc này.

Bài 2: Dự đoán từ tiếp theo (Next Token Prediction)

Sử dụng mô hình GPT (Decoder-only) để viết tiếp câu.

Để kết quả có thể tái lập (tùy chọn), bạn có thể set seed

```
from transformers import pipeline, set_seed

set_seed(42)
```

1. Tải pipeline "text-generation" (thường dùng GPT-2 mặc định)

```
generator = pipeline("text-generation", model="gpt2")
```


```
{"model_id":"99db537c40874200af85385cea9b22e8","version_major":2,"version_minor":0}

{"model_id":"eaa6736d8f1645a78d37d3ed0bd3a58c","version_major":2,"version_minor":0}

{"model_id":"198dc6eab02348159d2940bd4259ec15","version_major":2,"version_minor":0}

{"model_id":"1582a651f4f546cead16d0d4a2cb5ceb","version_major":2,"version_minor":0}

{"model_id":"4fd83a4d7d0f4afa8db73405ed18cab1","version_major":2,"version_minor":0}

{"model_id":"d022c3d5ccae474cb01d07d3535d35ce","version_major":2,"version_minor":0}

{"model_id":"2a925cba2d214ef78f8ac1633740e557","version_major":2,"version_minor":0}
```


```
Device set to use cpu
```

2. Đoạn văn bản mồi

```
prompt = "The best thing about learning NLP is"

prompt
```

```
{"type":"string"}
```

3. Sinh văn bản

```
# max_length: độ dài tối đa
# num_return_sequences: số lượng câu muốn sinh ra
generated_texts = generator(prompt, max_length=50,
num_return_sequences=1, truncation=True)
```

```
Setting `pad_token_id` to `eos_token_id`:50256 for open-end
generation.
Both `max_new_tokens` (=256) and `max_length`(=50) seem to have been
set. `max_new_tokens` will take precedence. Please refer to the
documentation for more information.
(https://huggingface.co/docs/transformers/main/en/main_classes/text_ge
neration)
```

4. In kết quả

```
print(f"Câu mồi: '{prompt}'")
print("-" * 30)
for i, text in enumerate(generated_texts):
    print(f"Văn bản sinh ra {i+1}:")
    print(text['generated_text'])
```

```
Câu mồi: 'The best thing about learning NLP is'
------------------------------
Văn bản sinh ra 1:
The best thing about learning NLP is that there are a few ways to pick
up resources:

Learn something from others.

Learning from teachers.

Learning from friends.

Learning from friends who are knowledgeable and articulate.

Learning from family.

Learning from friends who have had a strong interest in learning NLP.

I wish I had found a good way to give back. I have done this, I have
done this, I have done this, I have done this, I have done this, I
```

```
have done this.

If you're looking for something new, or you look to learn something
from other people, then this is the place to start. I encourage people
to do this, and to share their experiences with others.

So when you're ready to start learning NLP, this is a great place to
start.

What's next?

I'm always looking for the next big thing. I've been in the top 10 (in
our top 10 list for NLP learning), my top 10 list for NLP learning,
I'm going to start this year with a book.

I'm already working on this. It could be a book, I'm just going to
focus on the book for now.
```

Trả lời câu hỏi Bài 2:

1, Kết quả sinh ra có hợp lý không?

=> Trả lời: Khá hợp lý về mặt ngữ pháp, tuy nhiên ý nghĩa có thể hơi ngẫu nhiên do bản chất của việc sinh văn bản tự động (Generative AI).

2, Tại sao các mô hình Decoder-only như GPT lại phù hợp cho tác vụ này?

=> Trả lời: Vì Decoder hoạt động theo cơ chế tự hồi quy (autoregressive) và nhìn một chiều (unidirectional). Nó được huấn luyện để dự đoán từ tiếp theo dựa trên chuỗi các từ đã xuất hiện trước đó, giống như cách con người viết văn bản tuần tự từ trái sang phải.

4. Bài 3: Tính toán Vector biểu diễn của câu (Sentence Representation)

Lấy vector đặc trưng (embedding) của câu bằng phương pháp Mean Pooling từ mô hình BERT.

1. Chọn mô hình BERT base

```python
import torch
from transformers import AutoTokenizer, AutoModel

model_name = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModel.from_pretrained(model_name)
```

{"model_id":"001646c7f4eb4cc8a11071eb239770a1","version_major":2,"version_minor":0}

{"model_id":"5b8733f7791d4c5b9675479490e7f009","version_major":2,"version_minor":0}

{"model_id":"ed7093608bc44f4abdb560ac09d418e9","version_major":2,"version_minor":0}

{"model_id":"ef6a18752fb84438ad1ec1c3071bf943","version_major":2,"version_minor":0}

{"model_id":"189bc82731bc4155823ad6fd384d9ebd","version_major":2,"version_minor":0}

2. Câu đầu vào

```
sentences = ["This is a sample sentence."]

sentences
```

```
['This is a sample sentence.']
```

3. Tokenize câu

```
# return_tensors='pt': trả về PyTorch tensors
inputs = tokenizer(sentences, padding=True, truncation=True,
return_tensors='pt')

inputs
```

```
{'input_ids': tensor([[ 101, 2023, 2003, 1037, 7099, 6251, 1012,
102]]), 'token_type_ids': tensor([[0, 0, 0, 0, 0, 0, 0, 0]]),
'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1]])}
```

4. Đưa qua mô hình để lấy hidden states

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

# Lấy hidden state cuối cùng (batch_size, sequence_length,
hidden_size)
last_hidden_state = outputs.last_hidden_state

last_hidden_state
```

```
tensor([[[-0.1993, -0.2101, -0.1950,  ..., -0.4733,  0.0861,  0.7103],
         [-0.5400, -0.7178, -0.2873,  ..., -0.7211,  0.5801,  0.3946],
         [-0.1421, -0.7375,  0.3737,  ..., -0.3740,  0.0750,  0.9687],
         ...,
         [ 0.1321, -0.2893, -0.0043,  ..., -0.1772, -0.2123, -0.1983],
```

```
         [ 0.4060,  0.0366, -0.7327,  ...,  0.4169, -0.3416, -0.4542],
         [ 0.0646, -0.2088, -0.1323,  ...,  0.5954, -1.0679,
0.0173]]])
```

5. Thực hiện Mean Pooling (Tính trung bình cộng có trọng số mask)

```
attention_mask = inputs['attention_mask']

attention_mask

tensor([[1, 1, 1, 1, 1, 1, 1, 1]])

# Mở rộng kích thước của mask để khớp với kích thước của hidden
state
# Mask gốc: [Batch, Seq_len] -> Mở rộng: [Batch, Seq_len,
Hidden_size]
mask_expanded = attention_mask.unsqueeze(-
1).expand(last_hidden_state.size()).float()

# Nhân hidden state với mask (để triệt tiêu giá trị của các token
padding về 0)
sum_embeddings = torch.sum(last_hidden_state * mask_expanded, 1)

# Tính tổng số token thực (không phải padding) để chia trung bình
# clamp(min=1e-9) để tránh lỗi chia cho 0
sum_mask = torch.clamp(mask_expanded.sum(1), min=1e-9)

# Tính trung bình
sentence_embedding = sum_embeddings / sum_mask
```

6. In kết quả

```
print("Vector biểu diễn của câu (5 giá trị đầu tiên):")
print(sentence_embedding[0][:5]) # Chỉ in 5 giá trị đầu để dễ nhìn
print("\nKích thước của vector:", sentence_embedding.shape)

Vector biểu diễn của câu (5 giá trị đầu tiên):
tensor([-0.0639, -0.4284, -0.0668, -0.3843, -0.0658])

Kích thước của vector: torch.Size([1, 768])
```

Trả lời câu hỏi Bài 3:

1, Kích thước (chiều) của vector biểu diễn là bao nhiêu? Con số này tương ứng với tham số nào?

=> Trả lời: Kích thước là (1, 768) (nếu batch size là 1). Chiều dài 768 tương ứng với tham số hidden_size (đôi khi gọi là d_model) của kiến trúc bert-base.

2, Tại sao chúng ta cần sử dụng attention_mask khi thực hiện Mean Pooling?

=> Trả lời: Để đảm bảo tính chính xác của phép tính trung bình.

- Khi xử lý theo batch, các câu ngắn hơn được thêm các token đệm (padding tokens, thường là số 0) để bằng độ dài câu dài nhất.
- Các token đệm này không mang ý nghĩa ngữ nghĩa.
- Nếu tính trung bình cộng tất cả các token mà không dùng mask, các giá trị của token đệm sẽ làm "loãng" hoặc sai lệch vector biểu diễn của câu. attention_mask giúp mô hình bỏ qua (nhân với 0) các token đệm này.