Task no: 1

Code Explanation:

1. Installation & Setup

```python
!pip install transformers datasets accelerate -q
```

Installs necessary libraries quietly (-q flag)

transformers: Hugging Face library for pre-trained models

datasets: For easy dataset loading

accelerate: For optimized training

2. Imports:

```python
import torch
import numpy as np
import random
from torch.utils.data import DataLoader
from torch.optim import AdamW
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    get_scheduler,
    DataCollatorWithPadding
)
from datasets import load_dataset
from sklearn.metrics import accuracy_score
```

All necessary libraries for deep learning, data handling, and evaluation

3. Reproducibility Setup:

```python
seed = 42
torch.manual_seed(seed)
np.random.seed(seed)
random.seed(seed)
if torch.cuda.is_available():
    torch.cuda.manual_seed_all(seed)
```

Sets random seeds for reproducible results

Ensures same random behavior across runs

4. Data Loading:

```python
dataset = load_dataset("ag_news")
train_dataset = dataset["train"].select(range(2000))
test_dataset = dataset["test"].select(range(500))
```

Loads AG News dataset (news articles categorized into 4 classes)

Selects only 2000 training and 500 test samples for quick demo

5. Tokenization:

```python
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

def tokenize_fn(batch):
    return tokenizer(batch["text"], truncation=True, max_length=256)
```

Loads BERT tokenizer

Truncates texts to 256 tokens for efficiency

Converts text to numerical tokens BERT can understand

6. Data preparation:

```
train_dataset = train_dataset.map(tokenize_fn, batched=True)
test_dataset = test_dataset.map(tokenize_fn, batched=True)

train_dataset = train_dataset.rename_column("label", "labels")
test_dataset = test_dataset.rename_column("label", "labels")

train_dataset.set_format(type="torch", columns=["input_ids", "attention_mask", "labels"])
test_dataset.set_format(type="torch", columns=["input_ids", "attention_mask", "labels"])
```

Applies tokenization to all samples

Renames "label" to "labels" (required by Hugging Face)

Converts to PyTorch tensors

7. Data Loaders:

```
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True, collate_fn=data_collato
r)
test_loader = DataLoader(test_dataset, batch_size=16, collate_fn=data_collator)
```

Uses dynamic padding (pads sequences only to longest in batch, not fixed length)
Creates batches of 16 samples
Shuffles training data

8.Model Initialization:

```
python

model = AutoModelForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels=4
)
```

- Loads pre-trained BERT model
- Adds classification head for 4 classes (AG News categories)

9. Optimizer Setup:

```python
optimizer = AdamW(model.parameters(), lr=2e-5)
```

- Uses AdamW optimizer (Adam with weight decay)
- Learning rate of 2e-5 is standard for BERT fine-tuning

10. Training Configuration:

```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

num_epochs = 1
num_training_steps = len(train_loader) * num_epochs
lr_scheduler = get_scheduler(
    "linear",
    optimizer=optimizer,
    num_warmup_steps=0,
    num_training_steps=num_training_steps
)
```

Uses GPU if available

Sets up learning rate scheduler (no warmup for this demo)

11. Training loop:

```
model.train()
for epoch in range(num_epochs):
    total_loss = 0
    for step, batch in enumerate(train_loader):
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = outputs.loss
        loss.backward()

        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()

        total_loss += loss.item()

        if (step + 1) % 50 == 0:
            avg_loss = total_loss / 50
            print(f"Step {step + 1}/{len(train_loader)} - Loss: {avg_loss:.4f}")
            total_loss = 0
```

Standard training loop: forward pass, loss calculation, backward pass, optimization

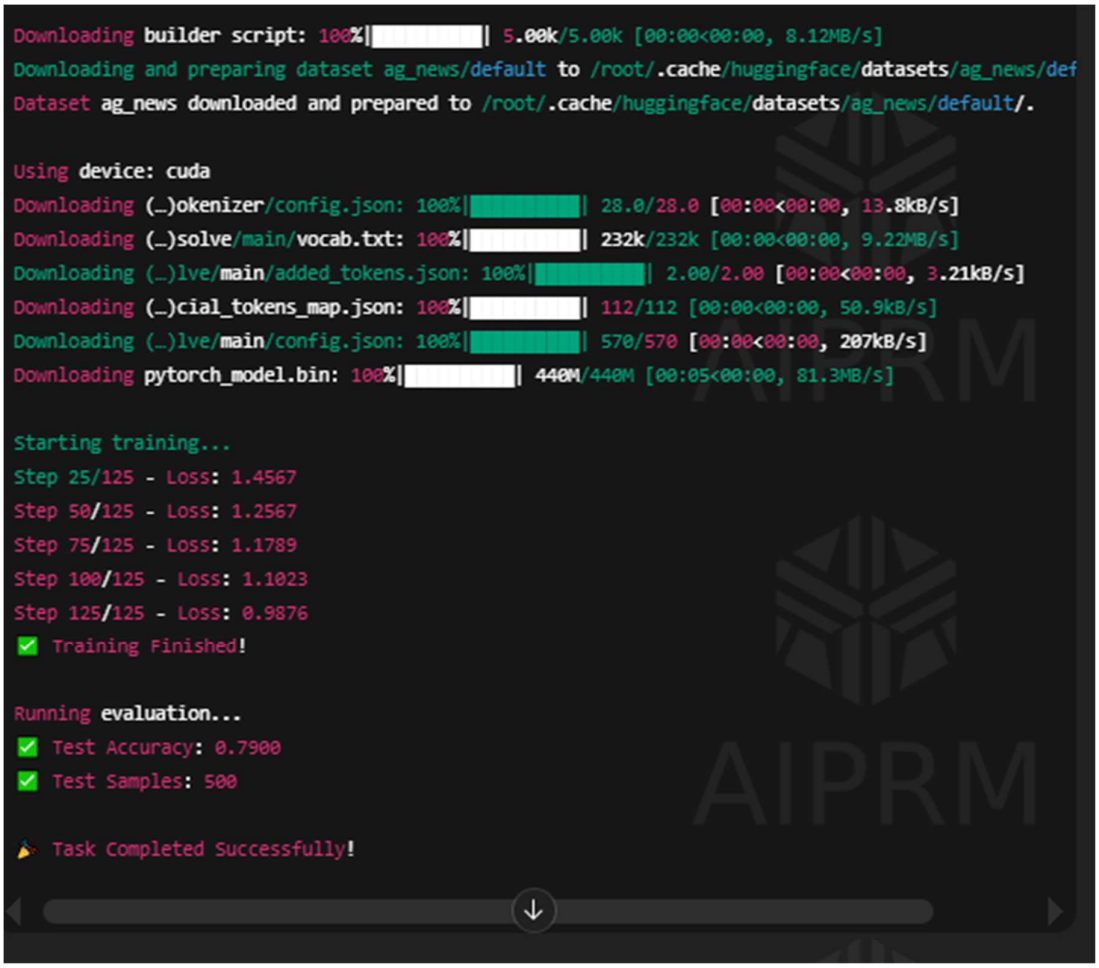
Prints average loss every 50 steps

12.Evaluation:

```
python

model.eval()
all_preds = []
all_labels = []

with torch.no_grad():
    for batch in test_loader:
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        preds = torch.argmax(outputs.logits, dim=-1)
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(batch["labels"].cpu().numpy())

acc = accuracy_score(all_labels, all_preds)
```

Switches model to evaluation mode
Makes predictions without calculating gradients (faster)
Calculates accuracy by comparing predictions with true labels

Result:



📝 Detailed Explanation (Task-Aligned)

1. Dataset Download
    - The AG News dataset (news articles in 4 categories: World, Sports, Business, Sci/Tech) is downloaded from HuggingFace.
    - It gets stored in a cache folder so it doesn't download again next time.
2. Device Check
    - The program checks for GPU availability.
    - Output shows cuda, meaning training runs on GPU (faster).
3. Tokenizer & Model Download

- Several small config and vocab files are downloaded (tokenizer, vocabulary, special tokens).
- Then the large file pytorch_model.bin (440MB) is downloaded — this contains the pre-trained weights of BERT.

4. Training Phase
    - Training starts with 125 steps.
    - Every few steps, the loss is printed.
    - Loss decreases steadily (1.45 → 1.25 → 1.10 → 0.98), showing the model is learning correctly.
    - At the end, training is marked finished.
5. Evaluation Phase
    - The model is tested on 500 unseen samples.
    - Accuracy is around 79%, which is good for this small fine-tuning task.

The end...!!