CS 335: Introduction to Large Language Models

Fine-Tuning
Week 7

Dr Abdul Samad

Loading a dataset

- Hugging Face has multiple datasets in lots of different languages
- You can browse the datasets here

The Hugging Face Datasets library allows you to easily download and cache datasets

```python
from datasets import load_dataset

raw_datasets = load_dataset("glue", "mrpc")
raw_datasets
```

Loading the "MRPC" dataset

Output

we get a `DatasetDict` object

```
DatasetDict({
    train: Dataset({
        features: ['sentence1', 'sentence2', 'label', 'idx'],
        num_rows: 3668
    })
    validation: Dataset({
        features: ['sentence1', 'sentence2', 'label', 'idx'],
        num_rows: 408
    })
    test: Dataset({
        features: ['sentence1', 'sentence2', 'label', 'idx'],
        num_rows: 1725
    })
})
```

Training Set

Validation Set

Test Set

The Hugging Face Datasets library allows you to easily download and cache datasets

```python
raw_train_dataset = raw_datasets["train"]
raw_train_dataset[0]
```

Output

```
{'idx': 0,
 'label': 1,
 'sentence1': 'Amrozi accused his brother , whom he called " the witness " , of deliberately distorting his evidence .',
 'sentence2': 'Referring to him as only " the witness " , Amrozi accused his brother of deliberately distorting his evidence .'}
```

We can access each pair of sentences in our raw_datasets object by indexing, like with a dictionary

```python
raw_datasets["train"][:5]
```

Output

```
{'idx': [0, 1, 2, 3, 4],
 'label': [1, 0, 1, 0, 1],
 'sentence1': ['Amrozi accused his brother , whom he called " the witness " , of deliberately distorting his evidence .',
  "Yucaipa owned Dominick 's before selling the chain to Safeway in 1998 for $ 2.5 billion .",
  'They had published an advertisement on the Internet on June 10 , offering the cargo for sale , he added .',
  'Around 0335 GMT , Tab shares were up 19 cents , or 4.4 % , at A $ 4.56 , having earlier set a record high of A $ 4.57 .',
  'The stock rose $ 2.11 , or about 11 percent , to close Friday at $ 21.51 on the New York Stock Exchange .'],
 'sentence2': ['Referring to him as only " the witness " , Amrozi accused his brother of deliberately distorting his evidence .',
  "Yucaipa bought Dominick 's in 1995 for $ 693 million and sold it to Safeway for $ 1.8 billion in 1998 .",
  "On June 10 , the ship 's owners had published an advertisement on the Internet , offering the explosives for sale .",
  'Tab shares jumped 20 cents , or 4.6 % , to set a record closing high at A $ 4.57 .',
  'PG & E Corp. shares jumped $ 1.63 or 8 percent to $ 21.03 on the New York Stock Exchange on Friday .']}
```

You can also directly get a slice of your dataset

```python
raw_datasets["train"].features
```

Output

```
{'sentence1': Value(dtype='string', id=None),
 'sentence2': Value(dtype='string', id=None),
 'label': ClassLabel(num_classes=2, names=['not_equivalent', 'equivalent'], names_file=None, id=None),
 'idx': Value(dtype='int32', id=None)}
```

Label ID: 0

Label ID: 1

The features attributes gives us more information about each column

Preprocessing a dataset

```python
from transformers import AutoTokenizer

checkpoint = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
sequences = [
    "I've been waiting for a HuggingFace course my whole life.",
    "This course is amazing!",
]
batch = tokenizer(sequences, padding=True, truncation=True, return_tensors="pt")
print(batch)
```

Output

```
{'input_ids': tensor([[  101,  1045,  1005,  2310,  2042,  3403,  2005,  1037, 17662, 12172,
2607,  2026,  2878,  2166,  1012,   102],
[  101,  2023,  2607,  2003,  6429,   999,   102,     0,     0,     0,
0,     0,     0,     0,     0,     0]]),
'token_type_ids': tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],[0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0]]),
'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],[1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0]])}
```

Tokenize single sentences and batch them together

“What are the best resources for learning morse code?”
“What is morse code?”

Not Duplicate

“How does an IQ test work and what is determined from an IQ test?”
“How does IQ test work?”

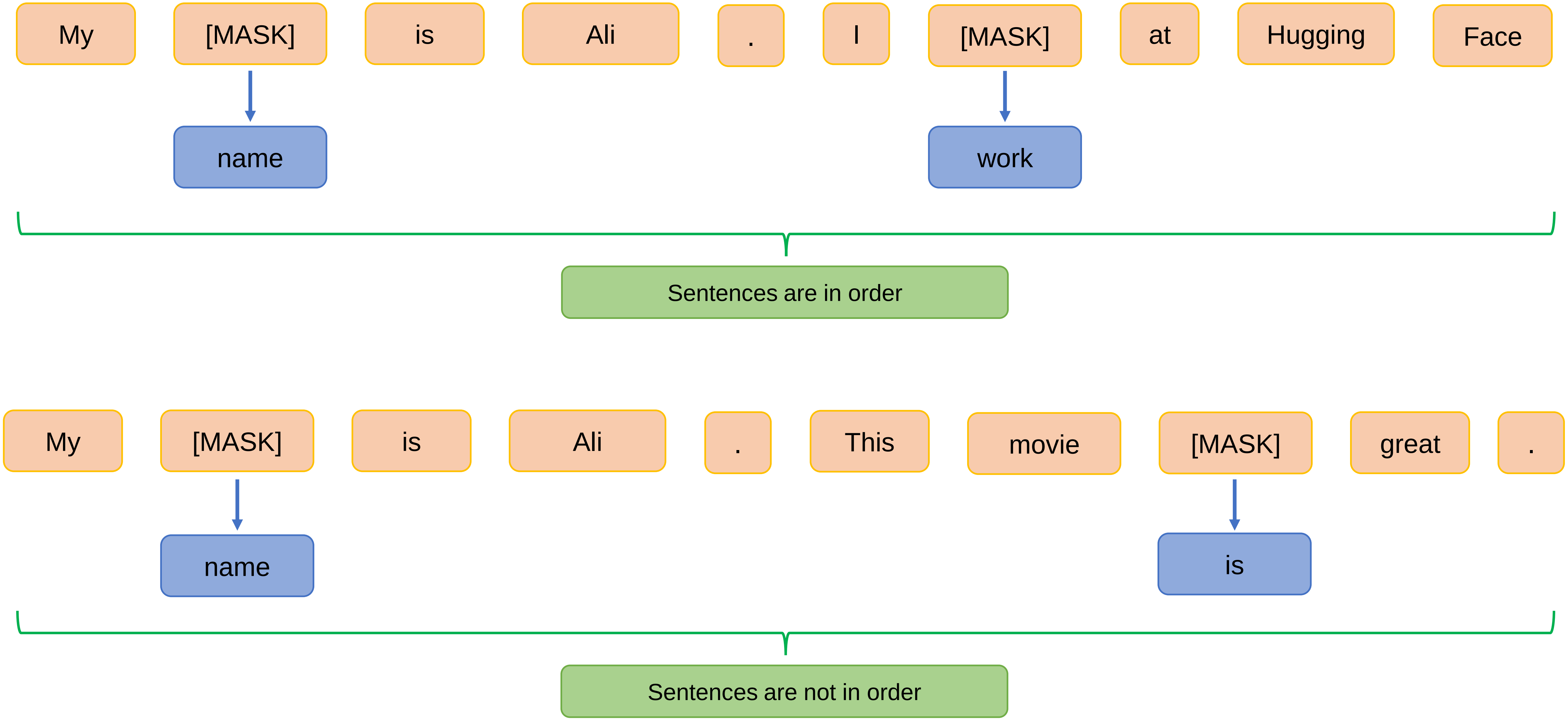
Duplicate

Are these questions duplicates or not

Text classification can also be applied on pairs of sentences

Sentence pair	Label
“What are the best resources for learning morse code?” “What is morse code?”	Not Duplicate
“How does an IQ test work and what is determined from an IQ test?” “How does IQ test work?”	Duplicate
“Fun for only children.” “Fun for adults and children.”	contradiction
“Well you’re a mechanics student right?” “yeah well you’re student right”	neutral
“The other men were shuffled around” “The other men shuffled.”	entailment

Text classification can also be applied on pairs of sentences



Models like BERT are pretrained to recognize relationships between two sentences

```python
from transformers import AutoTokenizer

checkpoint = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
tokenizer("My name is Ali.", "I work at Hugging Face.")
```

Output

```
{
  'input_ids': [101, 2023, 2003, 1996, 2034, 6251, 1012, 102, 2023, 2003, 1996, 2117, 2028, 1012, 102],
  'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
  'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
}
```

The tokenizers accept sentence pairs as well as single sentences

```
{
  'input_ids': [101, 2023, 2003, 1996, 2034, 6251, 1012, 102, 2023, 2003, 1996, 2117, 2028, 1012, 102],
  'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
  'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
}
```

ID	101	2026	2171	2003	25353	22144	2378	1012	102	1045	2147	2012	17662	2227	1012	102
token	[CLS]	my	name	is	sy	##lva	##in	.	[SEP]	I	work	at	hugging	face	.	[SEP]
Token type	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
attention	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

As you can see, the parts of the input corresponding to [CLS] sentence1 [SEP] will have a token type ID of 0, while the other parts, corresponding to sentence2 [SEP], all have a token type ID of 1.

The tokenizer adds special tokens for the corresponding model and prepares token type IDs to indicate which part of the inputs correspond to which sentence

```python
from transformers import AutoTokenizer

checkpoint = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
tokenizer(
    ["My name is Abdul.", "Going to the cinema."],
    ["I work at Hugging Face.", "This movie is great."],
    padding=True
)
```

Output

Sentence 1

```
{'input_ids': [
    [101, 2026, 2171, 2003, 25353, 22144, 2378, 1012, 102, 1045, 2147, 2012, 17662, 2227, 1012, 102],
    [101, 2183, 2000, 1996, 5988, 1012, 102, 2023, 3185, 2003, 2307, 1012, 102, 0, 0, 0]
    ],
 'token_type_ids': [
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1],
    [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0]
    ],
 'attention_mask': [
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0]
    ]
}
```

Sentence 2

To process several pairs of sentences together, just pass the list of first sentences followed by the list of second sentences

- This is the first sentence
- This is the second one. It is longer
- The third one is even longer. It has many words.
- This one is short.

This	is	the	first	sent	##ence	.	[PAD]	[PAD]	[PAD]	[PAD]	[PAD]
This	is	the	second	one	.	It	is	longer	.	[PAD]	[PAD]
The	third	one	is	even	longer	.	It	has	many	words	.
This	one	is	short	.	[PAD]	[PAD]	[PAD]	[PAD]	[PAD]	[PAD]	[PAD]

We need to pad sentences of different lengths to make batches

- **Pro: All the batches will have the same shape**
- **Con: Lots of batches will have useless columns with pad tokens only**

Length of the longest sentence in the whole dataset

This	is	the	first	sent	##ence	.	[PAD]	[PAD]	[PAD]	[PAD]	[PAD]
This	is	the	second	one	.	It	is	longer	.	[PAD]	[PAD]
The	third	one	is	even	longer	.	It	has	many	words	.
This	one	is	short	.	[PAD]	[PAD]	[PAD]	[PAD]	[PAD]	[PAD]	[PAD]

The first way to do this is to pad the sentences in the whole dataset to the maximum length in the dataset

Length of the longest sentence in the whole dataset

This	is	the	first	sent	##ence	.	[PAD]	[PAD]	[PAD]	[PAD]	[PAD]
This	is	the	second	one	.	It	is	longer	.	[PAD]	[PAD]
The	third	one	is	even	longer	.	It	has	many	words	.
This	one	is	short	.	[PAD]	[PAD]	[PAD]	[PAD]	[PAD]	[PAD]	[PAD]

Another way is to pad the sentences at the batch creation. To the length of the longest sentence; a technique called dynamic padding

```python
from datasets import load_dataset
from transformers import AutoTokenizer

raw_datasets = load_dataset("glue", "mrpc")
checkpoint = "bert-base-cased"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)

def tokenize_function(examples):
    return tokenizer(
        examples["sentence1"], examples["sentence2"], padding="max_length", truncation=True,
max_length=128
    )

tokenized_datasets = raw_datasets.map(tokenize_function, batched=True)
tokenized_datasets = tokenized_datasets.remove_columns(["idx", "sentence1", "sentence2"])
tokenized_datasets = tokenized_datasets.rename_column("label", "labels")
tokenized_datasets = tokenized_datasets.with_format("torch")
```

Preprocess dataset with fixed padding

```python
from datasets import load_dataset
from transformers import AutoTokenizer, DataCollatorWithPadding

raw_datasets = load_dataset("glue", "mrpc")
checkpoint = "bert-base-cased"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)

def tokenize_function(examples):
    return tokenizer(examples["sentence1"], examples["sentence2"], truncation=True)

tokenized_datasets = raw_datasets.map(tokenize_function, batched=True)
data_collator = DataCollatorWithPadding(tokenizer)
```

Preprocess dataset with dynamic padding

```python
from transformers import AutoTokenizer

checkpoint = "bert-base-cased"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)

def tokenize_function(example):
    return tokenizer(
        example["sentence1"], example["sentence2"], padding="max_length",
truncation=True, max_length=128
    )

tokenized_datasets = raw_datasets.map(tokenize_function)
print(tokenized_datasets.column_names)
```

Output

```
{'train': ['attention_mask', 'idx', 'input_ids', 'label', 'sentence1', 'sentence2', 'token_type_ids'],
'validation': ['attention_mask', 'idx', 'input_ids', 'label', 'sentence1', 'sentence2',
'token_type_ids'], 'test': ['attention_mask', 'idx', 'input_ids', 'label', 'sentence1', 'sentence2',
'token_type_ids']}
```

The map method allows you to apply a function over all the splits of a given dataset

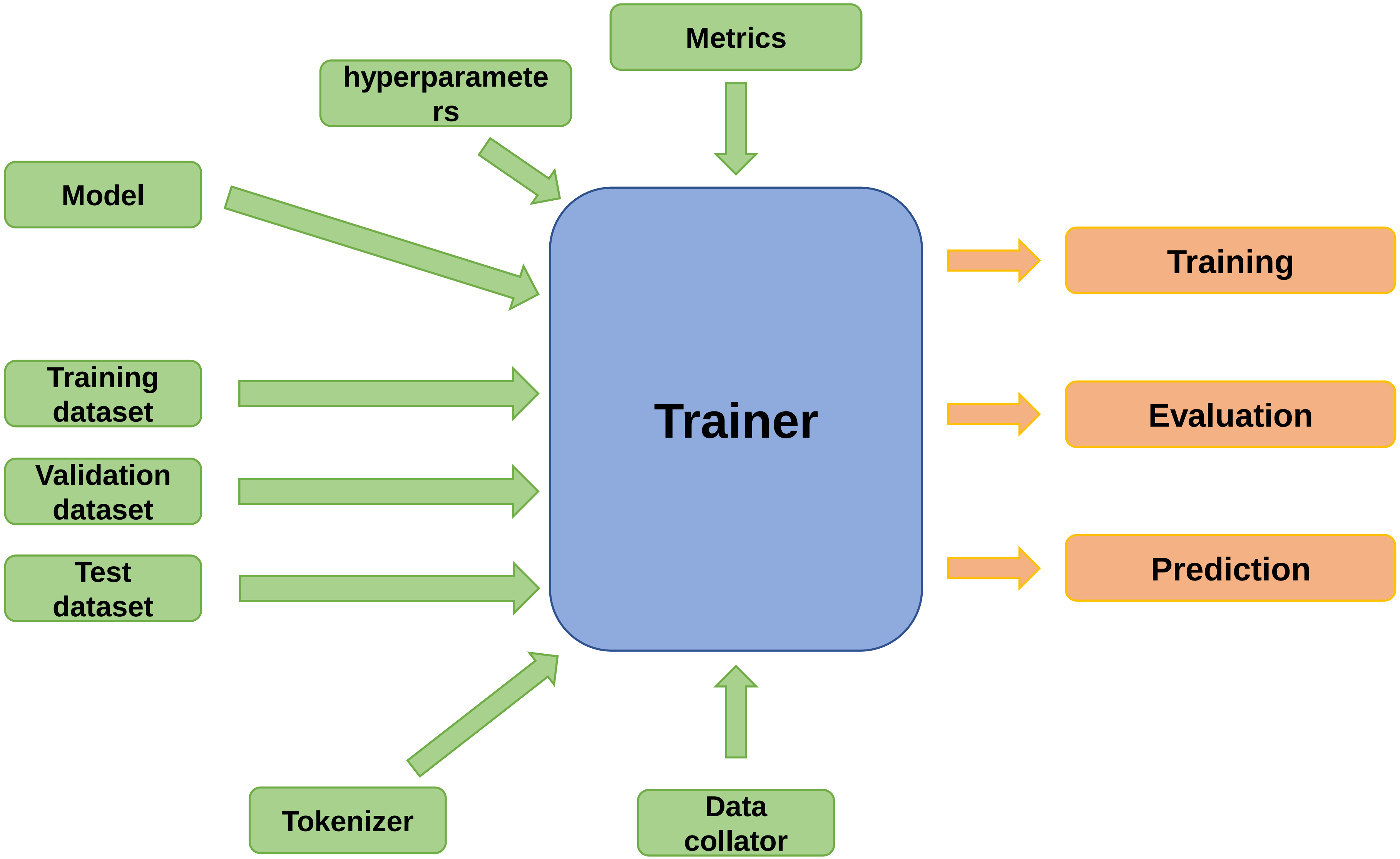
```python
from transformers import AutoTokenizer

checkpoint = "bert-base-cased"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)

def tokenize_function(example):
    return tokenizer(
        example["sentence1"], example["sentence2"], padding="max_length",
truncation=True, max_length=128
    )

tokenized_datasets = raw_datasets.map(tokenize_function, batched = True)
print(tokenized_datasets.column_names)
```

Output

```
{'train': ['attention_mask', 'idx', 'input_ids', 'label', 'sentence1', 'sentence2', 'token_type_ids'],
'validation': ['attention_mask', 'idx', 'input_ids', 'label', 'sentence1', 'sentence2',
'token_type_ids'], 'test': ['attention_mask', 'idx', 'input_ids', 'label', 'sentence1', 'sentence2',
'token_type_ids']}
```

You can preprocess faster by using the option batched=True. The applied function will then receive multiple examples at each call

The Trainer API



Transformers provide a Trainer API to easily train or fine-tune Transformer models

```python
from transformers import AutoModelForSequenceClassification

model = AutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels=2)
```

```python
from transformers import TrainingArguments

training_args = TrainingArguments("test-trainer")
```

We also need a model and some training arguments before creating the Trainer

```python
from transformers import AutoModelForSequenceClassification

model = AutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels=2)
```

```python
from transformers import TrainingArguments

training_args = TrainingArguments(
    "test-trainer",
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=5,
    learning_rate=2e-5,
    weight_decay=0.01,
)
```

We also need a model and some training arguments before creating the Trainer

```
from transformers import Trainer

trainer = Trainer(
    model,
    training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    data_collator=data_collator,
    tokenizer=tokenizer,
)
trainer.train()
```

We can then pass everything to the trainer class and start training

```python
predictions = trainer.predict(tokenized_datasets["validation"])
print(predictions.predictions.shape, predictions.label_ids.shape)
```

Output

```
(408, 2) (408,)
```

The predict method allows us to get the predictions of our model on a whole dataset. We can then use those predictions to compute metrics.

```python
predictions = trainer.predict(tokenized_datasets["validation"])
print(predictions.predictions.shape, predictions.label_ids.shape)
```

Output

```
(408, 2) (408,)
```

```python
import numpy as np
from datasets import load_metric

metric = load_metric("glue", "mrpc")
preds = np.argmax(predictions.predictions, axis=-1)
metric.compute(predictions=preds, references=predictions.label_ids)
```

Output

```
{
    'accuracy': 0.8627450980392157,
    'f1': 0.9050847457627118
}
```

You can preprocess faster by using the option batched=True. The applied function will then receive multiple examples at each call

```python
metric = load_metric("glue", "mrpc")

def compute_metrics(eval_preds):
    logits, labels = eval_preds
    predictions = np.argmax(logits, axis=-1)
    return metric.compute(predictions=predictions, references=labels)
```

```python
training_args = TrainingArguments("test-trainer", evaluation_strategy="epoch")
model = AutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels=2)

trainer = Trainer(
    model,
    training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["validation"],
    data_collator=data_collator,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)
trainer.train()
```

To monitor metrics during training we need to define a compute_metrics function and pass it to the Trainer

```
trainer.train()
```

Output

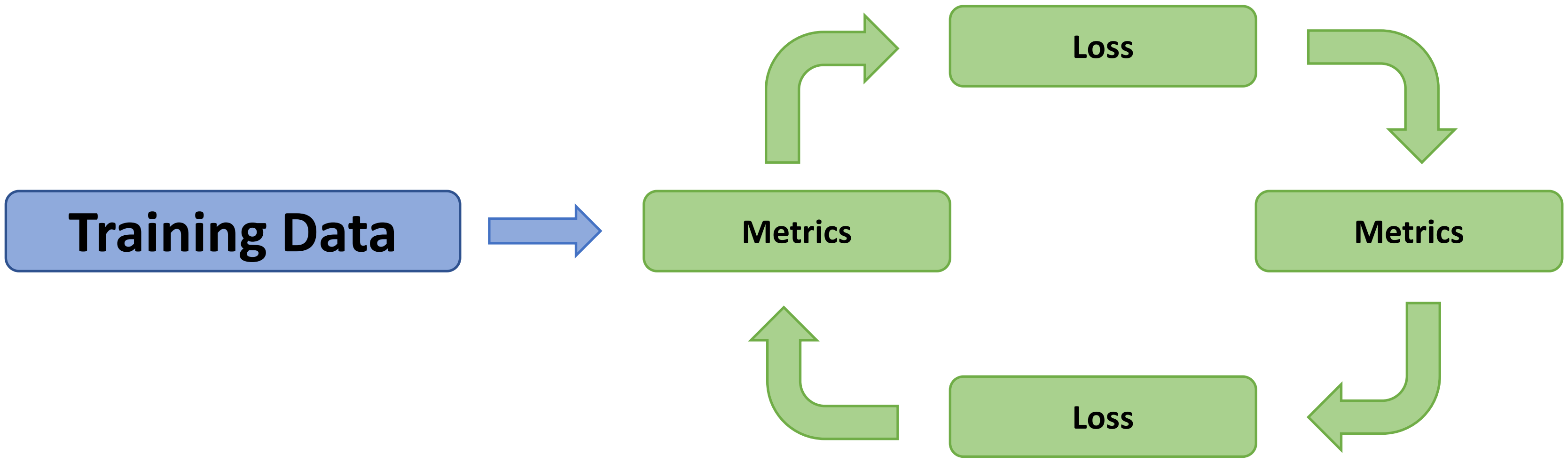
[1377/1377 03:56, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	F1
1	No log	0.402797	0.808824	0.865517
2	0.553000	0.423921	0.835784	0.887395
3	0.328400	0.646096	0.855392	0.899489

And we can launch a new training with metric reporting!

Training Loop

Splitting a raw text into words



A sketch of the training loop

```python
from torch.utils.data import DataLoader

train_dataloader = DataLoader(
  tokenized_datasets["train"], shuffle=True, batch_size=8, collate_fn=data_collator
)
eval_dataloader = DataLoader(
  tokenized_datasets["validation"], batch_size=8, collate_fn=data_collator
)
```

Once our data is preprocessed, we just have to create out DataLoaders

```python
from torch.utils.data import DataLoader

train_dataloader = DataLoader(
  tokenized_datasets["train"], shuffle=True, batch_size=8, collate_fn=data_collator
)
eval_dataloader = DataLoader(
  tokenized_datasets["validation"], batch_size=8, collate_fn=data_collator
)
```

```python
for batch in train_dataloader:
    break
print({k: v.shape for k, v in batch.items()})
```

Output

```
{'attention_mask': torch.Size([8, 63]), 'input_ids': torch.Size([8, 63]), 'labels': torch.Size([8]),
'token_type_ids': torch.Size([8, 63])}
```

Once our data is preprocessed, we just have to create out DataLoaders

```python
from transformers import AutoModelForSequenceClassification

checkpoint = "bert-base-cased"
model = AutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels=2)
```

The next step is to create our model

```python
from transformers import AutoModelForSequenceClassification

checkpoint = "bert-base-cased"
model = AutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels=2)
```

```python
outputs = model(**batch)
print(outputs.loss, outputs.logits.shape)
```

Output

```
tensor(0.7512, grad_fn=<NllLossBackward>) torch.Size([8, 2])
```

The next step is to create our model

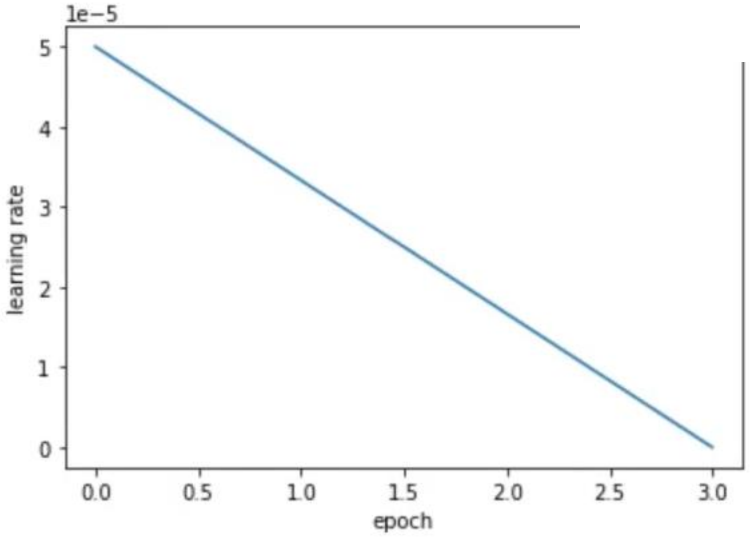
```python
from transformers import AdamW

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

```python
loss = outputs.loss
loss.backward()
optimizer.step()

# Don't forget to zero your gradients once your optimizer step is done!
optimizer.zero_grad()
```

The optimizer will be responsible for doing the training updates to the model weights

```python
from transformers import get_scheduler

num_epochs = 3
num_training_steps = num_epochs * len(train_dataloader)
lr_scheduler = get_scheduler(
    "linear",
    optimizer=optimizer,
    num_warmup_steps=0,
    num_training_steps=num_training_steps
)
```

Output



A learning rate scheduler will update the optimizer’s learning rate at each step

```python
import torch

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

Output

```
cuda
```

The training is going to run slowly if we don’t use a GPU

```python
from tqdm.auto import tqdm

progress_bar = tqdm(range(num_training_steps))

model.train()
for epoch in range(num_epochs):
    for batch in train_dataloader:
        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()
        progress_bar.update(1)
```

Putting everything together, here is what the training loop looks like

```python
from datasets import load_metric

metric= load_metric("glue", "mrpc")
model.eval()
for batch in eval_dataloader:
    batch = {k: v.to(device) for k, v in batch.items()}
    with torch.no_grad():
        outputs = model(**batch)

    logits = outputs.logits
    predictions = torch.argmax(logits, dim=-1)
    metric.add_batch(predictions=predictions, references=batch["labels"])

metric.compute()
```

Output

```
{
    'accuracy': 0.8284313725490197,
    'f1': 0.8809523809523808
}
```

Evaluation can be done like this with a Datasets metric

Supercharge your training loop with Accelerate

1 CPU

1 GPU

TPUs

Multiple CPUs

Multiple GPUs
one machine

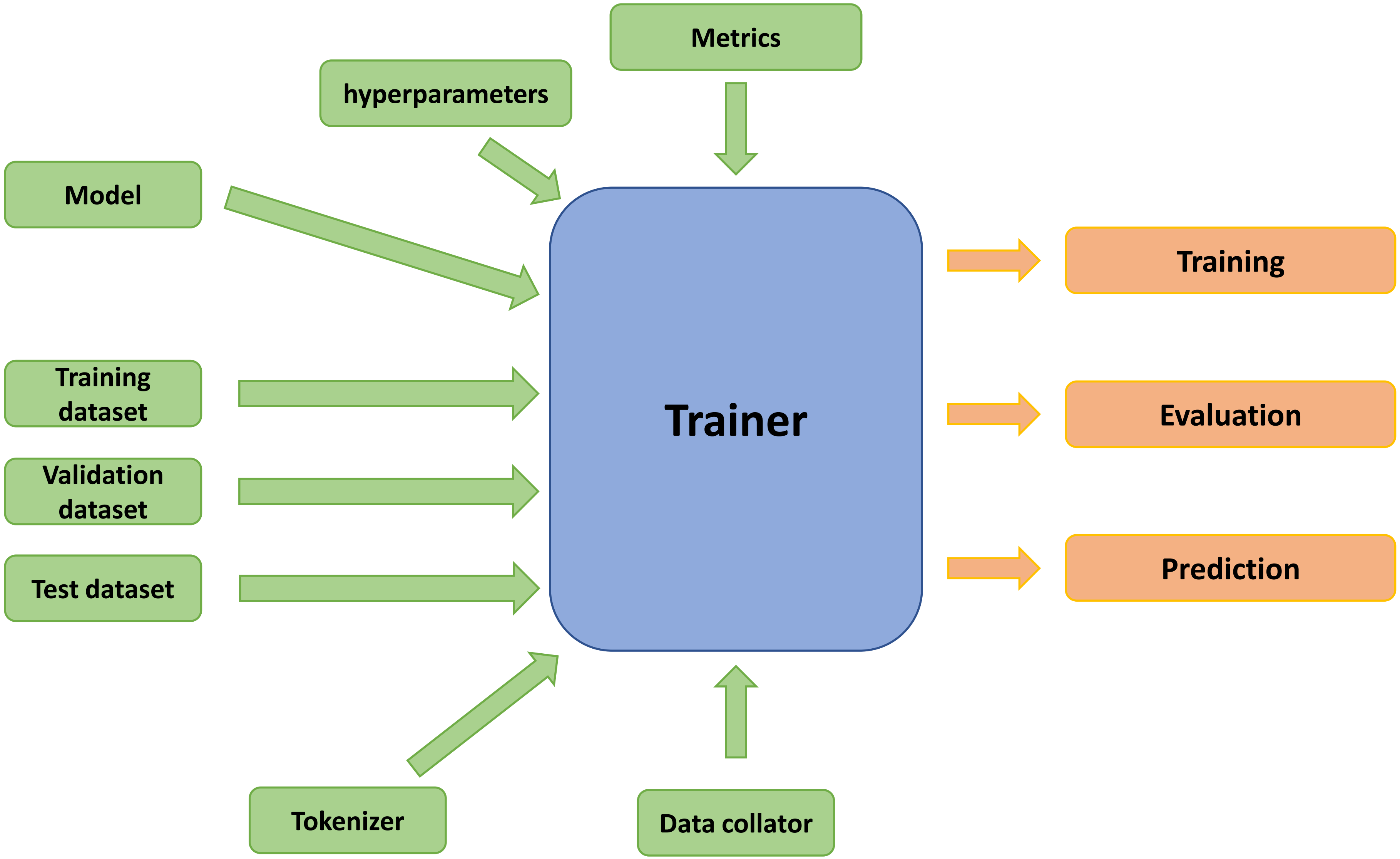
TPU pods

Multiple GPUs
Multiple machines

Trainer

Trainer

There are multiple setups on which you can run your training



The Trainer API can handle all those setups for you, but you lose control over your training loop

```python
from accelerate import Accelerator
from transformers import AdamW, AutoModelForSequenceClassification, get_scheduler

accelerator = Accelerator()

model = AutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels=2)
optimizer = AdamW(model.parameters(), lr=3e-5)

train_dl, eval_dl, model, optimizer = accelerator.prepare(
    train_dataloader, eval_dataloader, model, optimizer
)

num_epochs = 3
num_training_steps = num_epochs * len(train_dl)
lr_scheduler = get_scheduler(
    "linear",
    optimizer=optimizer,
    num_warmup_steps=0,
    num_training_steps=num_training_steps,
)

progress_bar = tqdm(range(num_training_steps))

model.train()
for epoch in range(num_epochs):
    for batch in train_dl:
        outputs = model(**batch)
        loss = outputs.loss
        accelerator.backward(loss)

        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()
        progress_bar.update(1)
```

Here’s what the complete training loop looks like with Accelerate