CS 696 Applied Large Language Models
Spring Semester, 2025
Doc 14 Finetuning 2
Feb 27, 2025

References

UCI Machine Learning Repository
https://archive.ics.uci.edu

Imbalance Learn
https://imbalanced-learn.org/stable/user_guide.html

Finetuning Large Language Models,
Sebastian Raschka, PhD
APR 22, 2023
https://magazine.sebastianraschka.com/p/finetuning-large-language-models

Building a Large Language Model (from Scratch), Sebastian Raschka

In Context Learning Guide

https://www.prompthub.us/blog/in-context-learning-guide#:~:text=In%2Dcontext%20Learning%20(ICL),prompt%2C%20known%20as%20%E2%80%9Ccontext.

Feature-Based Approach

Freeze the LLM

Run your data through the LLM

Use the output embeddings to train a different smaller model

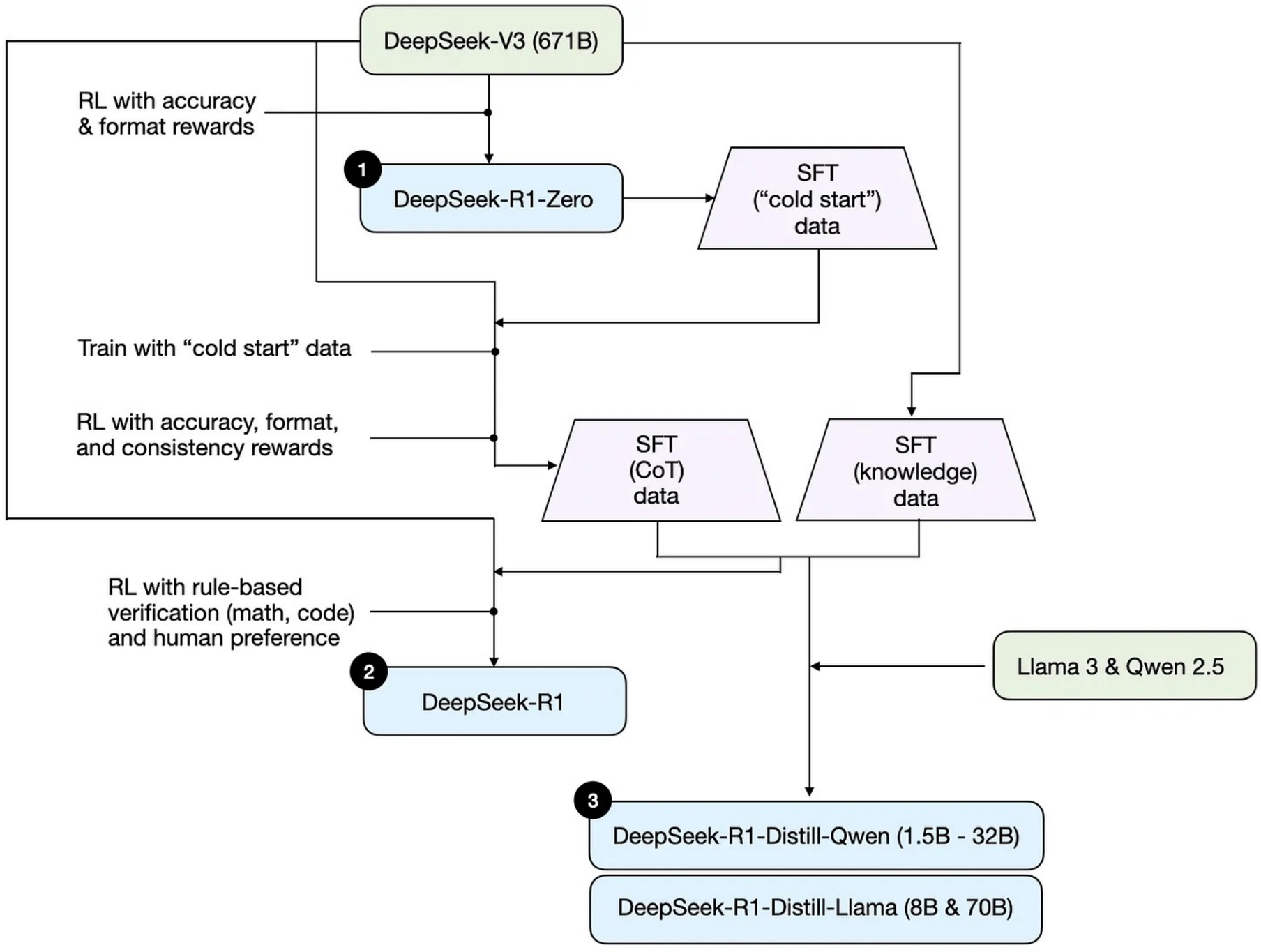
- LogisticRegression
- NN
- etc.

When to Use It:

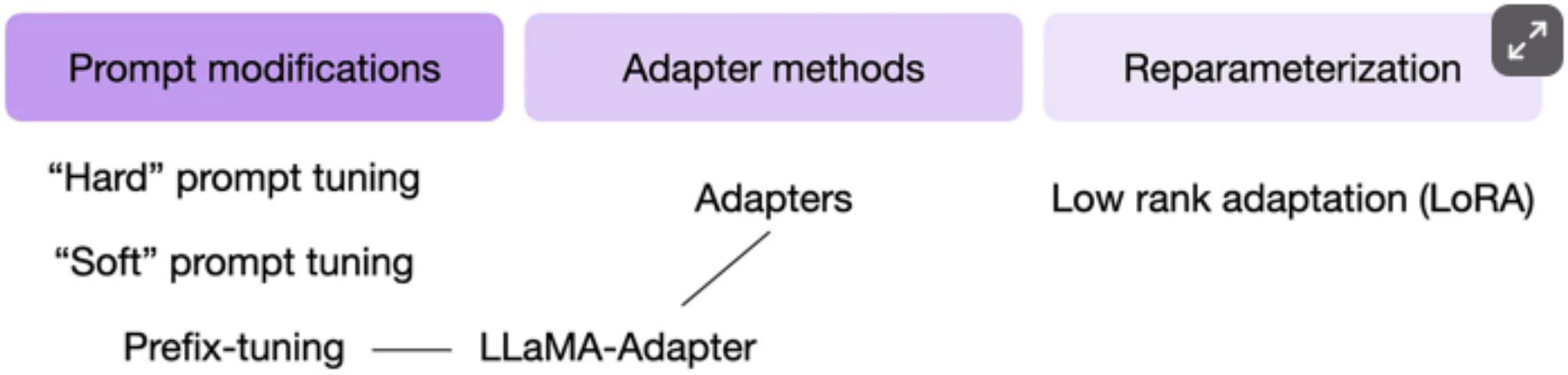
- When you have limited computational resources.
- When you have a small dataset.
- When you want to experiment with different downstream models.
- When you want to utilize a very large LLM, but do not have the resources to fine tune it.

https://github.com/rasbt/LLM-finetuning-scripts/blob/main/conventional/distilbert-movie-review/1_feature-extractor.ipynb

DeepSeek


DeepSeek-V3 (671B)
RL with accuracy & format rewards
1
DeepSeek-R1-Zero
SFT ("cold start") data
Train with "cold start" data
RL with accuracy, format, and consistency rewards
SFT (CoT) data
SFT (knowledge) data
RL with rule-based verification (math, code) and human preference
2
DeepSeek-R1
Llama 3 & Qwen 2.5
3
DeepSeek-R1-Distill-Qwen (1.5B - 32B)
DeepSeek-R1-Distill-Llama (8B & 70B)

Parameter-Efficient Finetuning



Low-Rank Adaptation (LoRA):

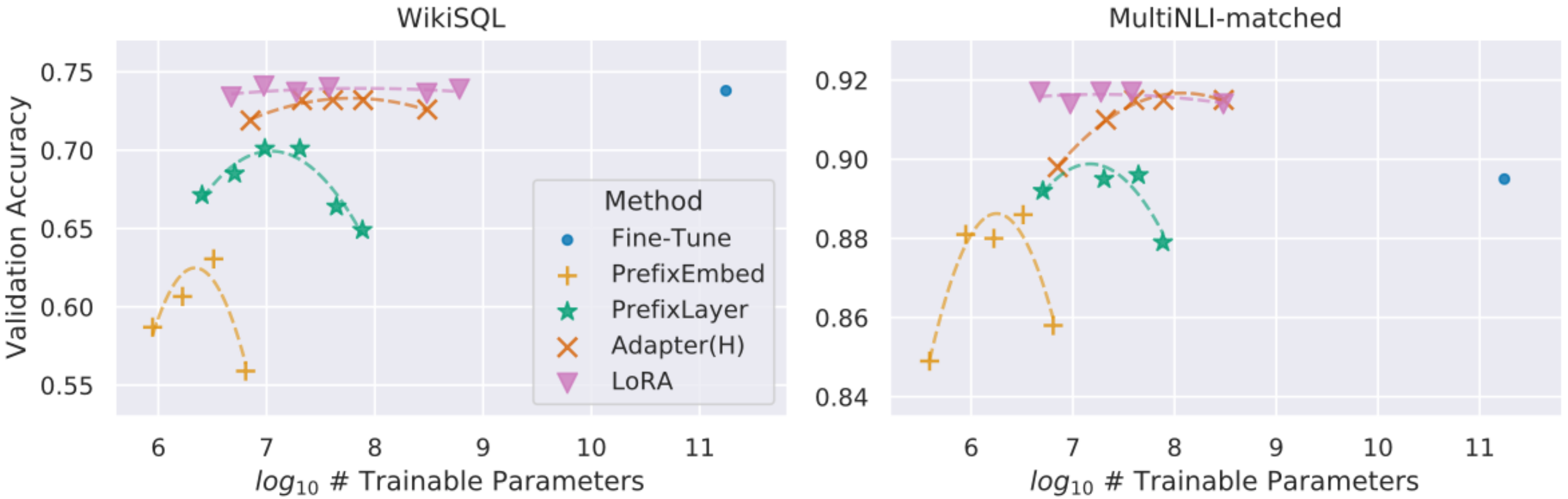
Adapter Layers:

Prompt Tuning:

Prefix Tuning:

QLoRA (Quantized Low-Rank Adapation):

LoRA - Low-Rank Adaptation Of LLMs



LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS

arXiv:2106.09685v2 [cs.CL] 16 Oct 2021

Matrix Rank

Maximum number of linearly independent rows or columns

A	B	C
0	I	I
I	3	5
3	2	8

C = 2* A + B

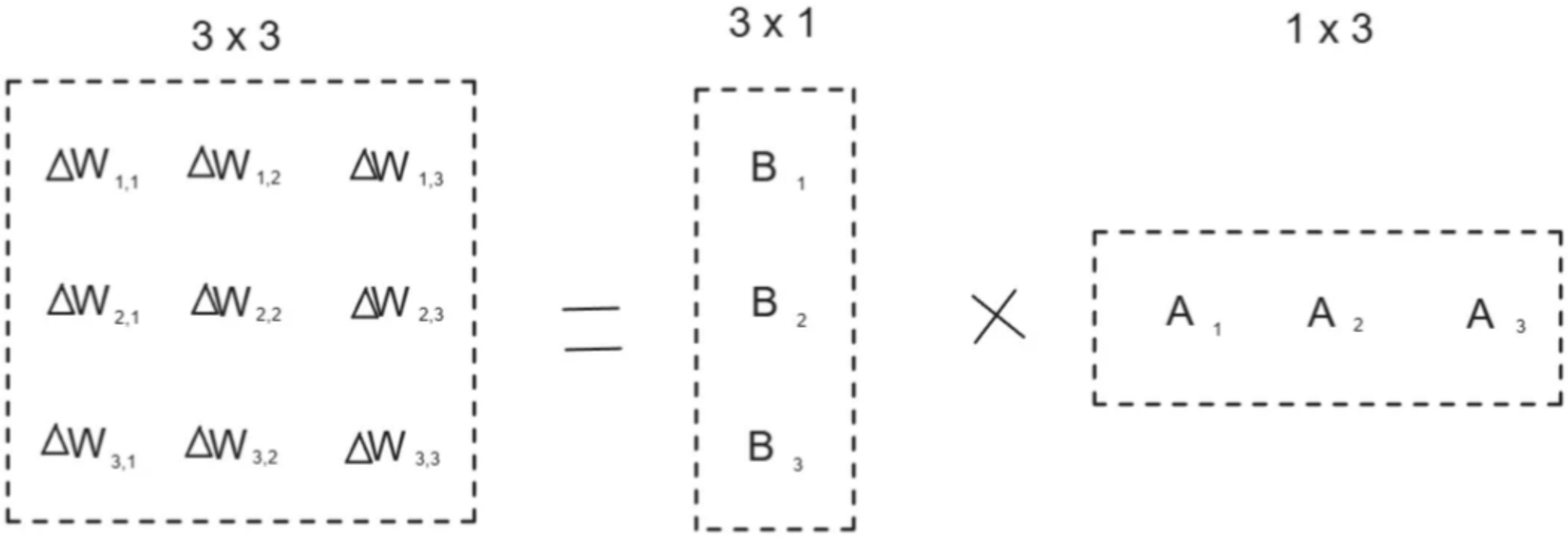
Rank 2

A	B
0	I
I	3
3	2

Basic Idea for LoRA

If the matrix has rank r, we can use a smaller matrix without losing much information

“When adapting to a specific task, Aghajanyan et al.(2020) shows that the pre-trained language models have a low intrinsic dimension”



LoRA Explained: Low-Rank Adaptation for Fine-Tuning LLMs, Zilliz

https://medium.com/@zilliz_learn/lora-explained-low-rank-adaptation-for-fine-tuning-llms-066c9bdd0b32

LoRA

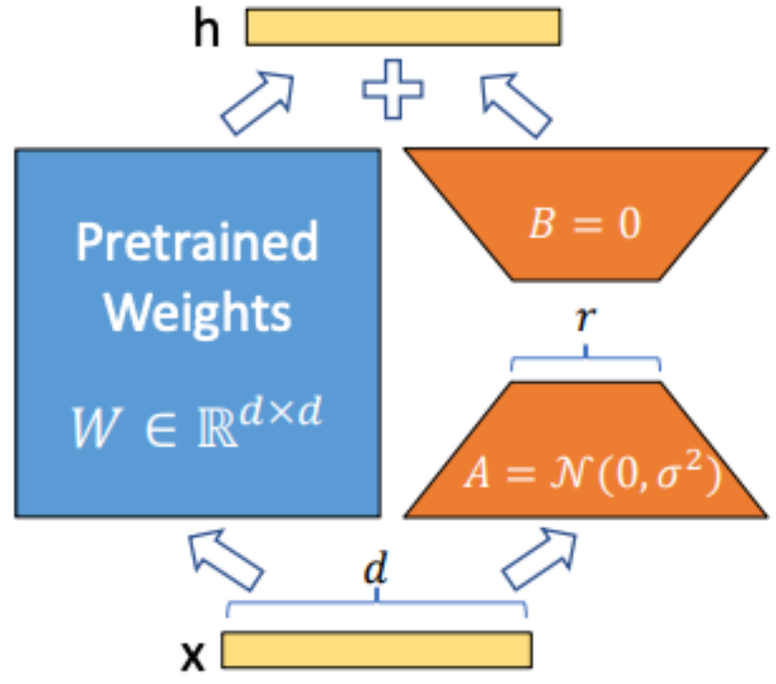
Freeze model weights

At each layer of the Transformer, add two matrices

Dimension dxr and rxd

Train on your fine-tune data

r is a hyperparameter



Model & Method	# Trainable Parameters	MNLI	SST-2	MRPC	CoLA	QNLI	QQP	RTE	STS-B	Avg.
RoB_{base} (FT)*	125.0M	**87.6**	94.8	90.2	**63.6**	92.8	**91.9**	78.7	91.2	86.4
RoB_{base} (BitFit)*	0.1M	84.7	93.7	**92.7**	62.0	91.8	84.0	81.5	90.8	85.2
RoB_{base} (Adpt^{D})*	0.3M	$87.1_{\pm.0}$	$94.2_{\pm.1}$	$88.5_{\pm1.1}$	$60.8_{\pm.4}$	$93.1_{\pm.1}$	$90.2_{\pm.0}$	$71.5_{\pm2.7}$	$89.7_{\pm.3}$	84.4
RoB_{base} (Adpt^{D})*	0.9M	$87.3_{\pm.1}$	$94.7_{\pm.3}$	$88.4_{\pm.1}$	$62.6_{\pm.9}$	$93.0_{\pm.2}$	$90.6_{\pm.0}$	$75.9_{\pm2.2}$	$90.3_{\pm.1}$	85.4
RoB_{base} (LoRA)	0.3M	$87.5_{\pm.3}$	$\mathbf{95.1}_{\pm.2}$	$89.7_{\pm.7}$	$63.4_{\pm1.2}$	$\mathbf{93.3}_{\pm.3}$	$90.8_{\pm.1}$	$\mathbf{86.6}_{\pm.7}$	$\mathbf{91.5}_{\pm.2}$	**87.2**
$\text{RoB}_{\text{large}}$ (FT)*	355.0M	90.2	**96.4**	**90.9**	68.0	94.7	**92.2**	86.6	92.4	88.9
$\text{RoB}_{\text{large}}$ (LoRA)	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.9}_{\pm1.2}$	$\mathbf{68.2}_{\pm1.9}$	$\mathbf{94.9}_{\pm.3}$	$91.6_{\pm.1}$	$\mathbf{87.4}_{\pm2.5}$	$\mathbf{92.6}_{\pm.2}$	**89.0**
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	3.0M	$90.2_{\pm.3}$	$96.1_{\pm.3}$	$90.2_{\pm.7}$	$\mathbf{68.3}_{\pm1.0}$	$\mathbf{94.8}_{\pm.2}$	$\mathbf{91.9}_{\pm.1}$	$83.8_{\pm2.9}$	$92.1_{\pm.7}$	88.4
$\text{RoB}_{\text{large}}$ (Adpt^{P})†	0.8M	$\mathbf{90.5}_{\pm.3}$	$\mathbf{96.6}_{\pm.2}$	$89.7_{\pm1.2}$	$67.8_{\pm2.5}$	$\mathbf{94.8}_{\pm.3}$	$91.7_{\pm.2}$	$80.1_{\pm2.9}$	$91.9_{\pm.4}$	87.9
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	6.0M	$89.9_{\pm.5}$	$96.2_{\pm.3}$	$88.7_{\pm2.9}$	$66.5_{\pm4.4}$	$94.7_{\pm.2}$	$92.1_{\pm.1}$	$83.4_{\pm1.1}$	$91.0_{\pm1.7}$	87.8
$\text{RoB}_{\text{large}}$ (Adpt^{H})†	0.8M	$90.3_{\pm.3}$	$96.3_{\pm.5}$	$87.7_{\pm1.7}$	$66.3_{\pm2.0}$	$94.7_{\pm.2}$	$91.5_{\pm.1}$	$72.9_{\pm2.9}$	$91.5_{\pm.5}$	86.4
$\text{RoB}_{\text{large}}$ (LoRA)†	0.8M	$\mathbf{90.6}_{\pm.2}$	$96.2_{\pm.5}$	$\mathbf{90.2}_{\pm1.0}$	$68.2_{\pm1.9}$	$\mathbf{94.8}_{\pm.3}$	$91.6_{\pm.2}$	$\mathbf{85.2}_{\pm1.1}$	$\mathbf{92.3}_{\pm.5}$	**88.6**
DeB_{XXL} (FT)*	1500.0M	91.8	**97.2**	92.0	72.0	**96.0**	92.7	93.9	92.9	91.1
DeB_{XXL} (LoRA)	4.7M	$\mathbf{91.9}_{\pm.2}$	$96.9_{\pm.2}$	$\mathbf{92.6}_{\pm.6}$	$\mathbf{72.4}_{\pm1.1}$	$\mathbf{96.0}_{\pm.1}$	$\mathbf{92.9}_{\pm.1}$	$\mathbf{94.9}_{\pm.4}$	$\mathbf{93.0}_{\pm.2}$	**91.3**

RoBERTabase, RoBERTalarge, and DeBERTaXXL

LORA: LOW-RANK ADAPTATION OF LARGE LANGUAGE MODELS

arXiv:2106.09685v2 [cs.CL] 16 Oct 2021

How do You Add Matrices to an existing Model?

PEFT

Python package from Huggingface

- PROMPT_TUNING
- MULTITASK_PROMPT_TUNING
- P_TUNING
- PREFIX_TUNING
- LORA
- ADALORA
- BOFT
- ADAPTION_PROMPT
- IA3
- LOHA
- LOKR
- OFT
- XLORA
- POLY
- LN_TUNING
- VERA
- FOURIERFT
- HRA
- BONE

But First Trainer

```
from datasets import load_dataset
from transformers import AutoTokenizer

dataset = load_dataset("yelp_review_full")

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

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

tokenized_datasets = dataset.map(tokenize_function, batched=True)

small_train_dataset = tokenized_datasets["train"].shuffle(seed=42).select(range(1000))
small_eval_dataset = tokenized_datasets["test"].shuffle(seed=42).select(range(1000))

from transformers import AutoModelForSequenceClassification

model = AutoModelForSequenceClassification.from_pretrained("google-bert/bert-base-cased",
num_labels=5, torch_dtype="auto")
```

But First Trainer

```
import numpy as np
import evaluate


metric = evaluate.load("accuracy")

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

```
from transformers import TrainingArguments, Trainer

training_args = TrainingArguments(output_dir="test_trainer", eval_strategy="epoch")

 trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=small_train_dataset,
    eval_dataset=small_eval_dataset,
    compute_metrics=compute_metrics,
 )
```

```
trainer.train()
```

```
Tracking run with wandb version 0.19.7
Run data is saved locally in /Users/rwhitney/Courses/696/Spring 2025 LLM/Notebooks/LoRA/wan
run-20250226_185730-cu61isqs
Syncing run test_trainer to Weights & Biases (docs)
View project at https://wandb.ai/whitney-san-diego-state-university/huggingface
View run at https://wandb.ai/whitney-san-diego-state-university/huggingface/runs/cu61isqs
```

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	1.149620	0.479000
2	No log	1.015054	0.564000
3	No log	1.048683	0.581000

```
TrainOutput(global_step=375,
    training_loss=1.040590576171875,
   metrics={'train_runtime': 507.8259, 'train_samples_per_second': 5.908,
       'train_steps_per_second': 0.738, 'total_flos': 78935442739200.0,
       ‘train_loss': 1.040590576171875, 'epoch': 3.0})
```

Wandb - Weights & Biases

Platform to help machine learning practitioners
    track,
    visualize, and
    collaborate on their experiment

Experiment Tracking

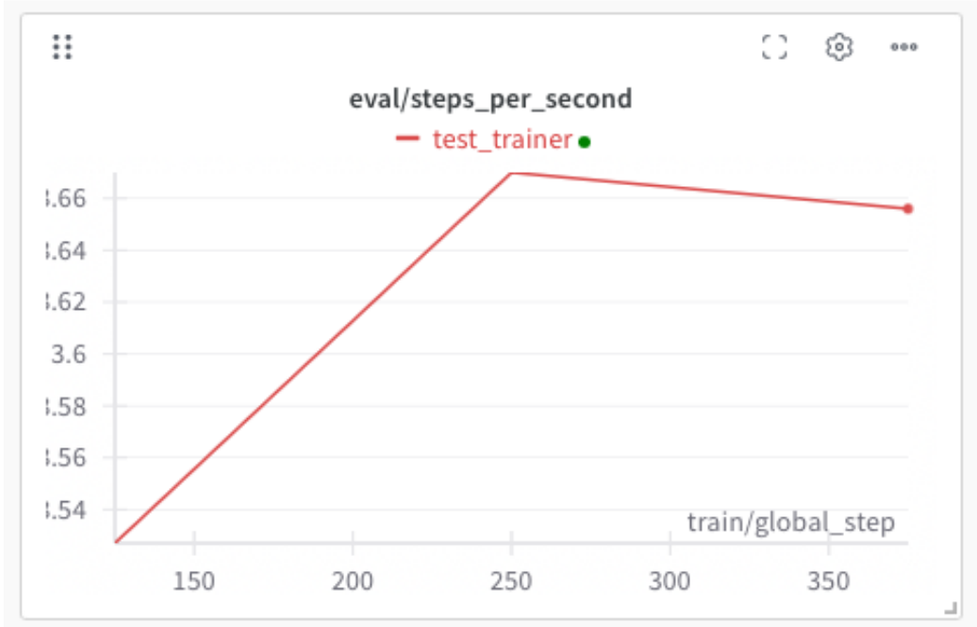
Visualization

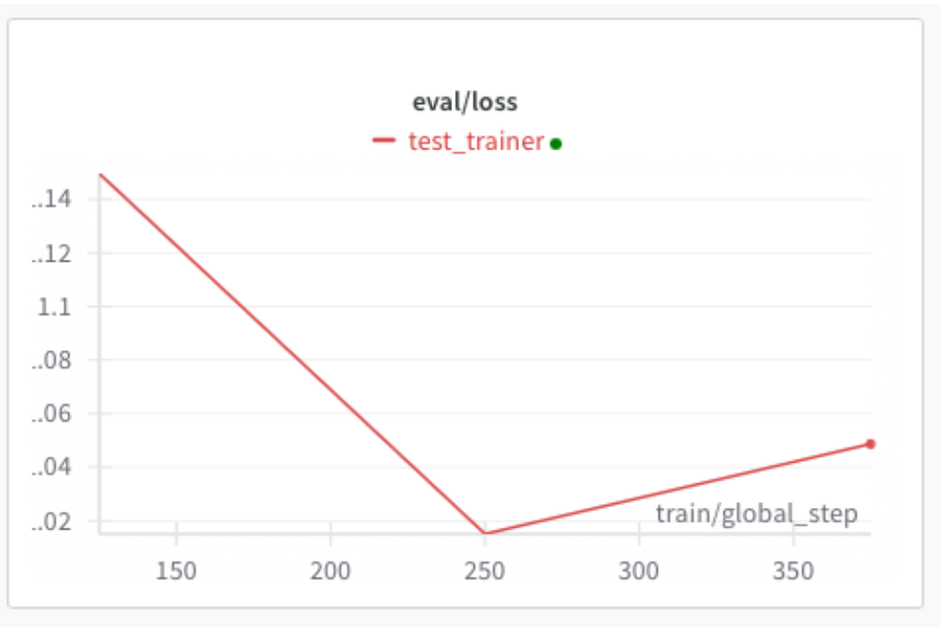
Collaboration

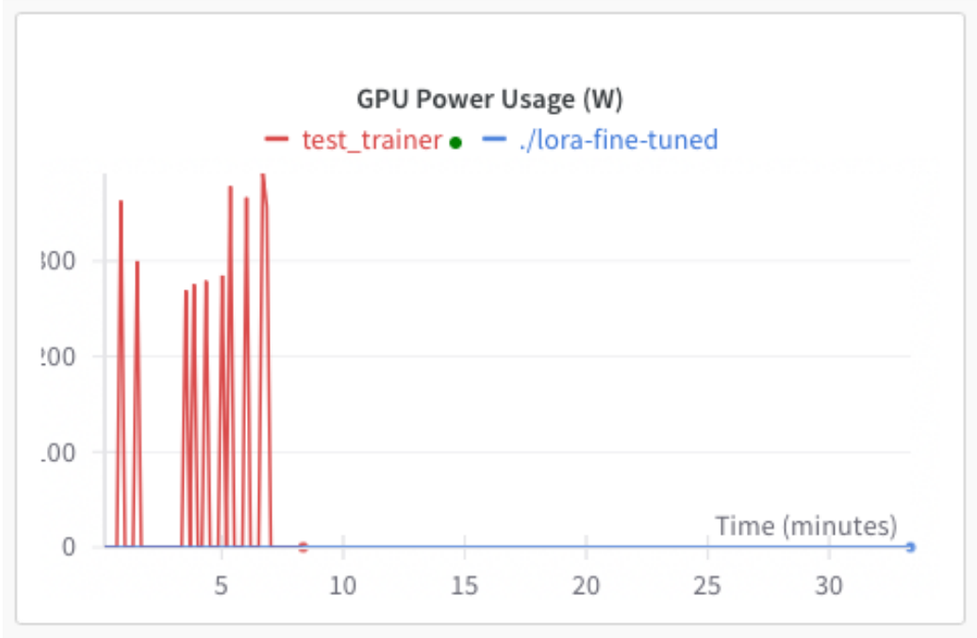
Model and Dataset Versioning

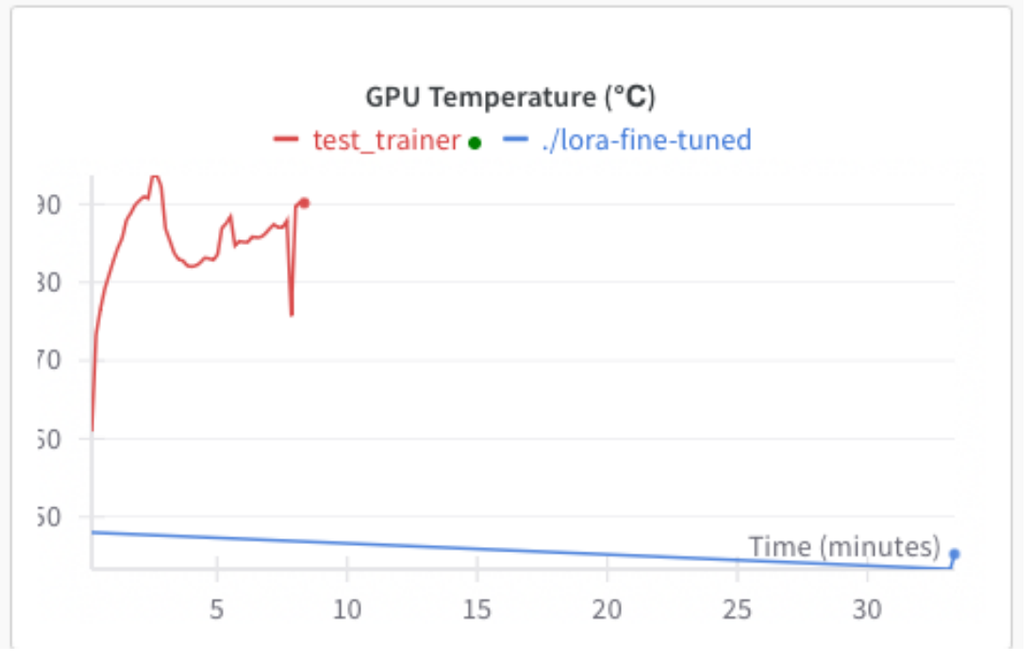
Tools for LLMs

Sample wandb.ai Graphs









Wandb.ai Weave

Log and debug language model inputs, outputs, and traces

pip install weave

```
import weave


@weave.op()
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return metric.compute(predictions=predictions, references=labels)

weave.init('huggingface')
```

compute_metrics `c8df`

Call | Code | Feedback | Scores | Summary | Use

Inputs

Path	Value
eval_pred	<transformers.trainer_utils.EvalPrediction object at 0x154…

Output

Path	Value
accuracy	0.529

Transformers Trainer

Simplified Training Loop

Perform a training step to calculate the loss
Calculate the gradients with the backward method
Update the weights based on the gradients
Repeat this process until you've reached a predetermined number of epochs

Built-in Evaluation and Logging:

Evaluation During Training - Valuation at specified intervals

Metrics Computation: Custom evaluation metrics

Logging

Checkpointing and Model Saving

Distributed Training Support

TrainingArguments

Arguments for Trainer

```
TrainingArguments(output_dir="test_trainer")
```

```
training_args = TrainingArguments(
    output_dir="your-model",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=2,
    weight_decay=0.01,
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    push_to_hub=True,
)
```

TrainingArguments - Output and Logging

output_dir:

The directory where training outputs (checkpoints, logs) will be saved.

logging_dir:

The directory where logs will be written.

logging_strategy:

Determines when logs are written (e.g., "steps," "epoch," “no”).

report_to:

Which services to report training logs to (e.g., "tensorboard," “wandb").

Etc

TrainingArguments - Training Parameters

num_train_epochs

per_device_train_batch_size

per_device_eval_batch_size

learning_rate

weight_decay

adam_beta1, adam_beta2, adam_epsilon

    Parameters for the AdamW optimizer.

max_grad_norm

warmup_steps

gradient_accumulation_steps

fp16

    Whether to use 16-bit (mixed) precision training.

evaluation_strategy

    Determines when evaluation is performed (e.g., "steps," "epoch," "no").

eval_steps

    How many update steps to wait between each evaluation.

TrainingArguments - Other

no_cuda

dataloader_num_workers

resume_from_checkpoint

push_to_hub

Customize the Trainer

Subclass Trainer methods

- get_train_dataloader()
- get_eval_dataloader()
- get_test_dataloader()
- log()
- create_optimizer()
- create_scheduler()
- compute_loss()
- training_step()
- prediction_step()
- evaluate()
- predict()

Optimizers

```
from transformers import TrainingArguments
training_args = TrainingArguments(..., optim="adamw_torch")
```

```
import torch

optimizer_cls = torch.optim.AdamW
optimizer_kwargs = {
    "lr": 4e-3,
    "betas": (0.9, 0.999),
    "weight_decay": 0.05,
}

from transformers import Trainer
trainer = Trainer(..., optimizer_cls_and_kwargs=(optimizer_cls, optimizer_kwargs))
```

Optimizers

```
optimizer = AdamW(model.parameters(), lr=2e-5, weight_decay=0.01)
scheduler = get_linear_schedule_with_warmup(
    optimizer, num_warmup_steps=500, num_training_steps=1000
)

trainer = Trainer(
    model=model,
    args=training_args,
    optimizers=(optimizer, scheduler),
```

Evaluate

Evaluation tools

- text,
- computer vision,
- audio,
- tools to evaluate models or dataset

Types of evaluations

Metric

Comparison

Measurement

```
import evaluate
accuracy = evaluate.load("accuracy")
```

```
accuracy.description
```

Accuracy is the proportion of correct predictions among the total number of cases processed. It can be computed with:

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Where:

TP: True positive

TN: True negative

FP: False positive

FN: False negative

```
accuracy.citation

  title={Scikit-learn: Machine Learning in {P}ython},
  author={Pedregosa, F. and Varoquaux, G. and Gramfort, A. and Michel, V.
         and Thirion, B. and Grisel, O. and Blondel, M. and Prettenhofer, P.
         and Weiss, R. and Dubourg, V. and Vanderplas, J. and Passos, A. and
         Cournapeau, D. and Brucher, M. and Perrot, M. and Duchesnay, E.},
  journal={Journal of Machine Learning Research},
  volume={12},
  pages={2825--2830},
  year={2011}
}

accuracy.features
{
    'predictions': Value(dtype='int32', id=None),
    'references': Value(dtype='int32', id=None)
}

accuracy.compute(references=[0,1,0,1], predictions=[1,0,0,1])
{'accuracy': 0.5}
```

But First Trainer

```
import numpy as np
import evaluate


metric = evaluate.load("accuracy")

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

```
from transformers import TrainingArguments, Trainer

training_args = TrainingArguments(output_dir="test_trainer", eval_strategy="epoch")

 trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=small_train_dataset,
    eval_dataset=small_eval_dataset,
    compute_metrics=compute_metrics,
 )
```

```
trainer.train()
```

```
Tracking run with wandb version 0.19.7
Run data is saved locally in /Users/rwhitney/Courses/696/Spring 2025 LLM/Notebooks/LoRA/wan
run-20250226_185730-cu61isqs
Syncing run test_trainer to Weights & Biases (docs)
View project at https://wandb.ai/whitney-san-diego-state-university/huggingface
View run at https://wandb.ai/whitney-san-diego-state-university/huggingface/runs/cu61isqs
```

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	1.149620	0.479000
2	No log	1.015054	0.564000
3	No log	1.048683	0.581000

```
TrainOutput(global_step=375,
    training_loss=1.040590576171875,
   metrics={'train_runtime': 507.8259, 'train_samples_per_second': 5.908,
       'train_steps_per_second': 0.738, 'total_flos': 78935442739200.0,
       ‘train_loss': 1.040590576171875, 'epoch': 3.0})
```

Example of Using LoRA

```
!pip install -q transformers
!pip install -q peft
!pip install -q evaluate

from datasets import load_dataset
from transformers import AutoTokenizer

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

    dataset = load_dataset("imdb")
    tokenizer = AutoTokenizer.from_pretrained("bert-base-cased")

    tokenized_datasets = dataset.map(tokenize_function, batched=True)

    ft_train = tokenized_datasets["train"].shuffle(seed=42).select(range(1000))
    ft_val = tokenized_datasets["test"].shuffle(seed=42).select(range(1000))
```

https://medium.com/@zilliz_learn/lora-explained-low-rank-adaptation-for-fine-tuning-llms-066c9bdd0b32

Adding the LoRA Matrices

```
from peft import LoraConfig, TaskType
from transformers import BertForSequenceClassification
from peft import get_peft_model

lora_config = LoraConfig(
    task_type=TaskType.SEQ_CLS, r=1, lora_alpha=1, lora_dropout=0.1
)

model = BertForSequenceClassification.from_pretrained(
    'bert-base-cased',
    num_labels=2
)

lora_model = get_peft_model(model, lora_config)
```

How Many Trainable Parameters

```
def count_parameters(model):
    trainable_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
    all_params = sum(p.numel() for p in model.parameters())
    trainable_percentage = 100 * trainable_params / all_params
    return trainable_params, all_params, trainable_percentage

trainable_params, all_params, trainable_percentage = count_parameters(lora_model)
print(f"trainable params: {trainable_params} || all params: {all_params} || trainable%: {trainable_percentage}")
```

```
trainable params:       38402
all params:         108350212
trainable%:                 0.035
```

```
import numpy as np
import evaluate
from transformers import TrainingArguments, Trainer

metric = evaluate.load(“accuracy")

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

```
training_args = TrainingArguments(output_dir="test_trainer", evaluation_strategy="epoch",
                                  num_train_epochs=25,)

trainer = Trainer(
    model=lora_model,
    args=training_args,
    train_dataset=ft_train,
    eval_dataset=ft_val,
    compute_metrics=compute_metrics,
)

trainer.train()
```

Output

Epoch	Training Loss	Validation Loss	Accuracy
1	No log	0.685166	0.580000
2	No log	0.679632	0.578000
3	No log	0.676504	0.606000
4	0.695000	0.674217	0.580000
5	0.695000	0.672407	0.615000

```
TrainOutput(global_step=625,
    training_loss=0.6929905395507813,
    metrics={'train_runtime': 712.7229,
                'train_samples_per_second': 7.015,
                 'train_steps_per_second': 0.877,
                'total_flos': 1316145131520000.0,
                'train_loss': 0.6929905395507813,
                ‘epoch': 5.0})
```

trainer.model

The Model

```
PeftModelForCausalLM(
  (base_model): LoraModel(
    (model): GPT2LMHeadModel(
      (transformer): GPT2Model(
        (wte): Embedding(50257, 768)
        (wpe): Embedding(1024, 768)
        (drop): Dropout(p=0.1, inplace=False)
        (h): ModuleList(
          (0-11): 12 x GPT2Block(
(ln_1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
(attn): GPT2Attention(
  (c_attn): lora.Linear(
    (base_layer): Conv1D(nf=2304, nx=768)
    (lora_dropout): ModuleDict(
      (default): Dropout(p=0.1, inplace=False)
    )
    (lora_A): ModuleDict(
      (default): Linear(in_features=768, out_features=8, bias=False)
    )
    (lora_B): ModuleDict(
      (default): Linear(in_features=8, out_features=2304, bias=False)
```

(0-11): 12 x GPT2Block(
(ln_1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
(attn): GPT2Attention(
(c_attn): lora.Linear(
(base_layer): Conv1D(nf=2304, nx=768)
(lora_dropout): ModuleDict(
(default): Dropout(p=0.1, inplace=False)
)
(lora_A): ModuleDict(
(default): Linear(in_features=768, **out_features=8**, bias=False)
)
(lora_B): ModuleDict(
(default): Linear(**in_features=8**, out_features=2304, bias=False)
)
(lora_embedding_A): ParameterDict()
(lora_embedding_B): ParameterDict()
(lora_magnitude_vector): ModuleDict()
)
(c_proj): Conv1D(nf=768, nx=768)
(attn_dropout): Dropout(p=0.1, inplace=False)
(resid_dropout): Dropout(p=0.1, inplace=False)
)
(ln_2): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
(mlp): GPT2MLP(
(c_fc): Conv1D(nf=3072, nx=768)
(c_proj): Conv1D(nf=768, nx=3072)
(act): NewGELUActivation()
(dropout): Dropout(p=0.1, inplace=False))))
(ln_f): LayerNorm((768,), eps=1e-05, elementwise_affine=True))
(lm_head): Linear(in_features=768, out_features=50257, bias=False))))

Checkpoints

Trainer saves molde checkpoints

```
# resume from latest checkpoint
trainer.train(resume_from_checkpoint=True)

# resume from specific checkpoint saved in output directory
trainer.train(resume_from_checkpoint="your-model/checkpoint-1000")
```

Hyperparameter Guidelines

Learning rate

1e-6 to larger values like 1e-3

Batch size

Common values being 1, 2, 4, 8, or 16

Larger: better gradient estimates but uses more memory

GPU 24 GB: batch 16

GPU 8 GB: Batch 2 or 4

Number of epochs

Typical range is 1 to 10 epochs

Large model, small dataset 1-3 epochs

Smaller model, larger dataset 5-10 epoch

Optimizers

AdamW

Weight decay

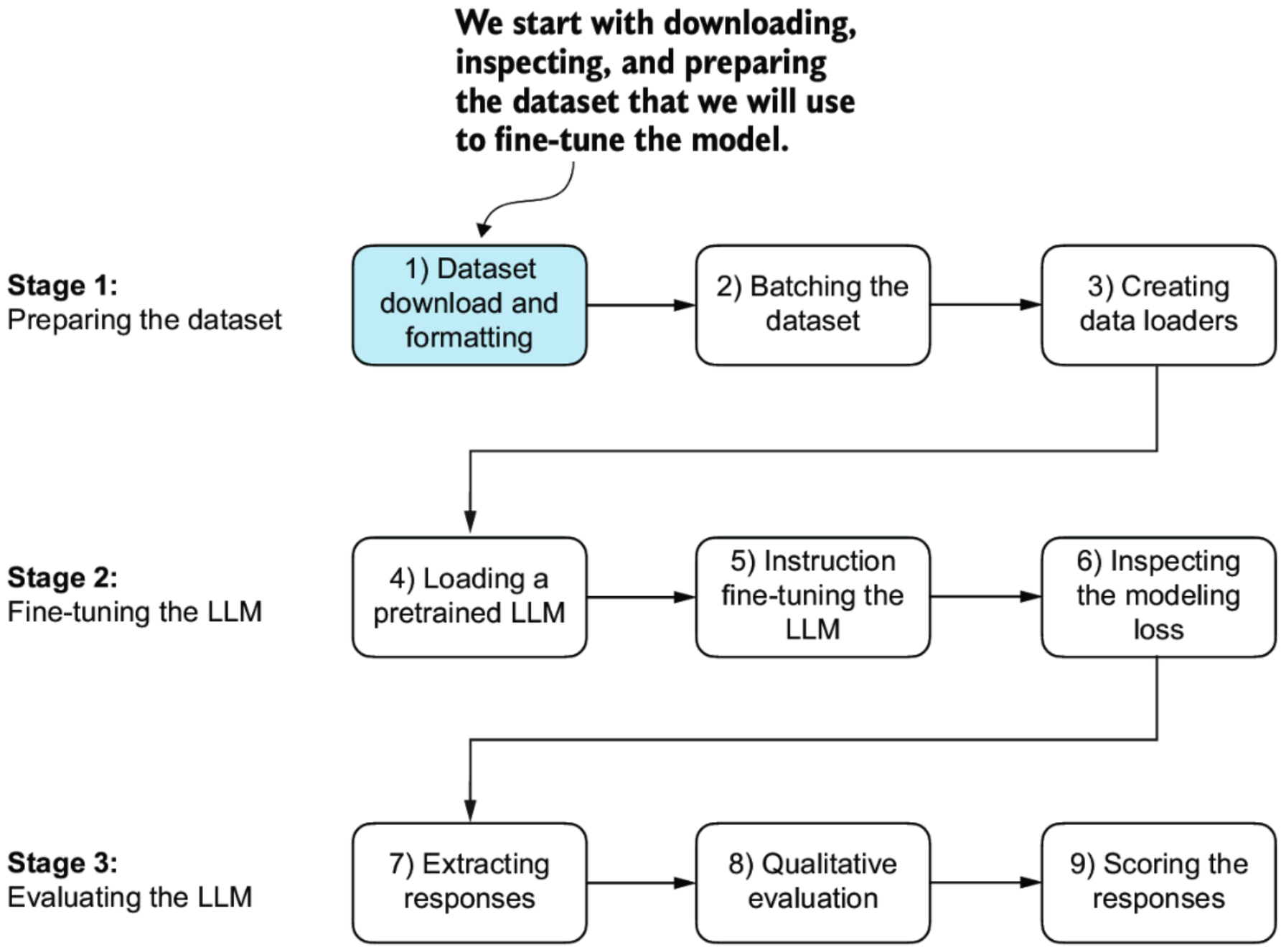
0.01 to 0.1,

0.01 common starting point

AdamW optimizer 0.01

LLM Engineer’s Handbook, Lusztin, Labonne

Instruction Fine-Tuning



An entry in the instruction dataset

```
{
    "instruction": "Identify the correct spelling of the following word.",
    "input": "Ocassion",
    "output": "The correct spelling is 'Occasion.'"
},
```

One way to format the data entry to train the LLM

Apply Alpaca prompt style template.

```
Below is an instruction that describes a task. Write a response that appropriately completes the request.

### Instruction:
Identify the correct spelling of the following word.

### Input:
Ocassion

### Response:
The correct spelling is 'Occasion'.
```

Apply Phi-3 prompt style template.

```
<|user|>
Identify the correct spelling of the following word: 'Ocassion'

<|assistant|>
The correct spelling is 'Occasion'.
```

Alpaca format

instruction: This is the instruction or prompt given to the language model.

input:

output:


```
{
  "instruction": "Write a short story about a cat who goes on an adventure.",
  "input": "",
  "output": "Whiskers twitched, Jasper the cat crept through the tall grass. He was on a mission, to find the legendary catnip patch..."
}


{
  "instruction": "human instruction (required)",
  "input": "human input (optional)",
  "output": "model response (required)",
  "system": "system prompt (optional)",
  "history": [
    ["human instruction in the first round (optional)", "model response in the first round (optional)"
    ["human instruction in the second round (optional)", "model response in the second round (optional)"]]}
```

Phi-3

<|system|>: Role for the model to assume. Ie. Python developer,

<|user|>: This is where you provide your actual prompt or question to the model.

<|assistant|>: This is where the model will generate its response.

<|system|> You are a helpful and informative AI assistant. <|end|>
<|user|> What are the main causes of climate change? <|end|>
<|assistant|>

Sample Data

{'instruction': 'Identify the correct spelling of the following word.',
'input': 'Ocassion',
‘output': "The correct spelling is 'Occasion.'"}

{'instruction': "What is an antonym of 'complicated'?",
'input': '',
‘output': "An antonym of 'complicated' is 'simple'."}

Formating

```
def format_input(entry):
    instruction_text = (
        f"Below is an instruction that describes a task. "
        f"Write a response that appropriately completes the request."
        f"\n\n### Instruction:\n{entry['instruction']}"
    )

    input_text = f"\n\n### Input:\n{entry['input']}" if entry["input"] else ""

    return instruction_text + input_text
```

Sample Formated Request

```
data = {'instruction': 'Identify the correct spelling of the following word.',
'input': 'Ocassion',
‘output': "The correct spelling is 'Occasion.'"}
```

```
model_input = format_input(data)
desired_response = f"\n\n### Response:\n{data['output']}"

print(model_input + desired_response)
```

```
Below is an instruction that describes a task. Write a response that appropriately completes the request.

### Instruction:
Identify the correct spelling of the following word.

### Input:
Ocassion

### Response:
The correct spelling is 'Occasion.'
```

InstructionDataset

```
import torch
from torch.utils.data import Dataset

class InstructionDataset(Dataset):
    def __init__(self, data, tokenizer):
        self.data = data

        self.encoded_texts = []
        for entry in data:
            instruction_plus_input = format_input(entry)
        response_text = f"\n\n### Response:\n{entry['output']}"
        full_text = instruction_plus_input + response_text
        self.encoded_texts.append(
            tokenizer.encode(full_text)
        )

def __getitem__(self, index):
    return self.encoded_texts[index]

def __len__(self):
    return len(self.data)
```

Preparing the Batchs

Tokenize data

Padding

Each batch can be a different length

Adding targets

Maximum length

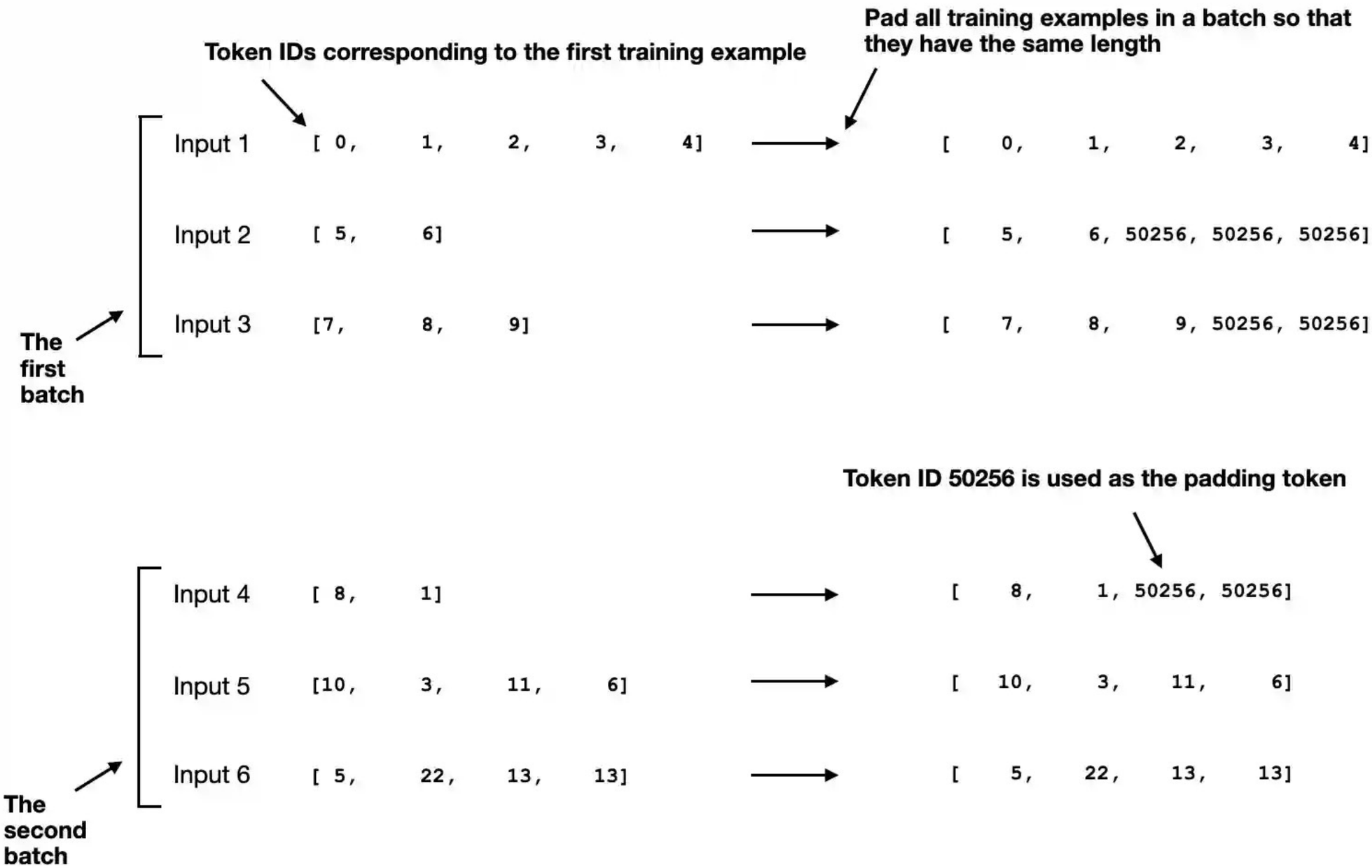
Ignore Index

So loss function ignores padding

cross_entropy ignores examples with label -100

Padding

Pad each patch individually so different lengths



Padding & Target

```
def custom_collate_draft_2(batch, pad_token_id=50256, device="cpu"):
    batch_max_length = max(len(item)+1 for item in batch)
    inputs_lst, targets_lst = [], []

    for item in batch:
        new_item = item.copy()
        new_item += [pad_token_id]
        # Pad sequences to max_length
        padded = (
            new_item + [pad_token_id] *
            (batch_max_length - len(new_item))
        )
        inputs = torch.tensor(padded[:-1])  # Truncate the last token for inputs
        targets = torch.tensor(padded[1:])  # Shift +1 to the right for targets
        inputs_lst.append(inputs)
        targets_lst.append(targets)

inputs_tensor = torch.stack(inputs_lst).to(device)
targets_tensor = torch.stack(targets_lst).to(device)
return inputs_tensor, targets_tensor
```

```
def custom_collate_fn(batch, pad_token_id=50256, ignore_index=-100,
        allowed_max_length=None, device="cpu"):

    batch_max_length = max(len(item)+1 for item in batch)
    inputs_lst, targets_lst = [], []

    for item in batch:
        new_item = item.copy()
        new_item += [pad_token_id]
        padded = (
            new_item + [pad_token_id] * (batch_max_length - len(new_item))
        )
        inputs = torch.tensor(padded[:-1])  # Truncate the last token for inputs
        targets = torch.tensor(padded[1:])  # Shift +1 to the right for targets

        # New: Replace all but the first padding tokens in targets by ignore_index
        mask = targets == pad_token_id
        indices = torch.nonzero(mask).squeeze()
        if indices.numel() > 1:
            targets[indices[1:]] = ignore_index

        if allowed_max_length is not None:
            inputs = inputs[:allowed_max_length]
            targets = targets[:allowed_max_length]

        inputs_lst.append(inputs)
        targets_lst.append(targets)

    inputs_tensor = torch.stack(inputs_lst).to(device)
    targets_tensor = torch.stack(targets_lst).to(device)
    return inputs_tensor, targets_tensor
```

```
inputs_1 = [0, 1, 2, 3, 4]
inputs_2 = [5, 6]
inputs_3 = [7, 8, 9]

batch = (
    inputs_1,
    inputs_2,
    inputs_3
)
```

```
inputs, targets = custom_collate_fn(batch)
print(inputs)
print(targets)
```

```
tensor([[    0,     1,     2,     3,     4],
        [    5,     6, 50256, 50256, 50256],
        [    7,     8,     9, 50256, 50256]])
tensor([[    1,     2,     3,     4, 50256],
        [    6, 50256,  -100,  -100,  -100],
        [    8,     9, 50256,  -100,  -100]])
```

Some Functional Magic

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

from functools import partial

customized_collate_fn = partial(
    custom_collate_fn,
    device=device,
    allowed_max_length=1024
)


customized_collate_fn(batch) =
    custom_collate_fn(batch, allowed_max_length=1024, device=device):
```

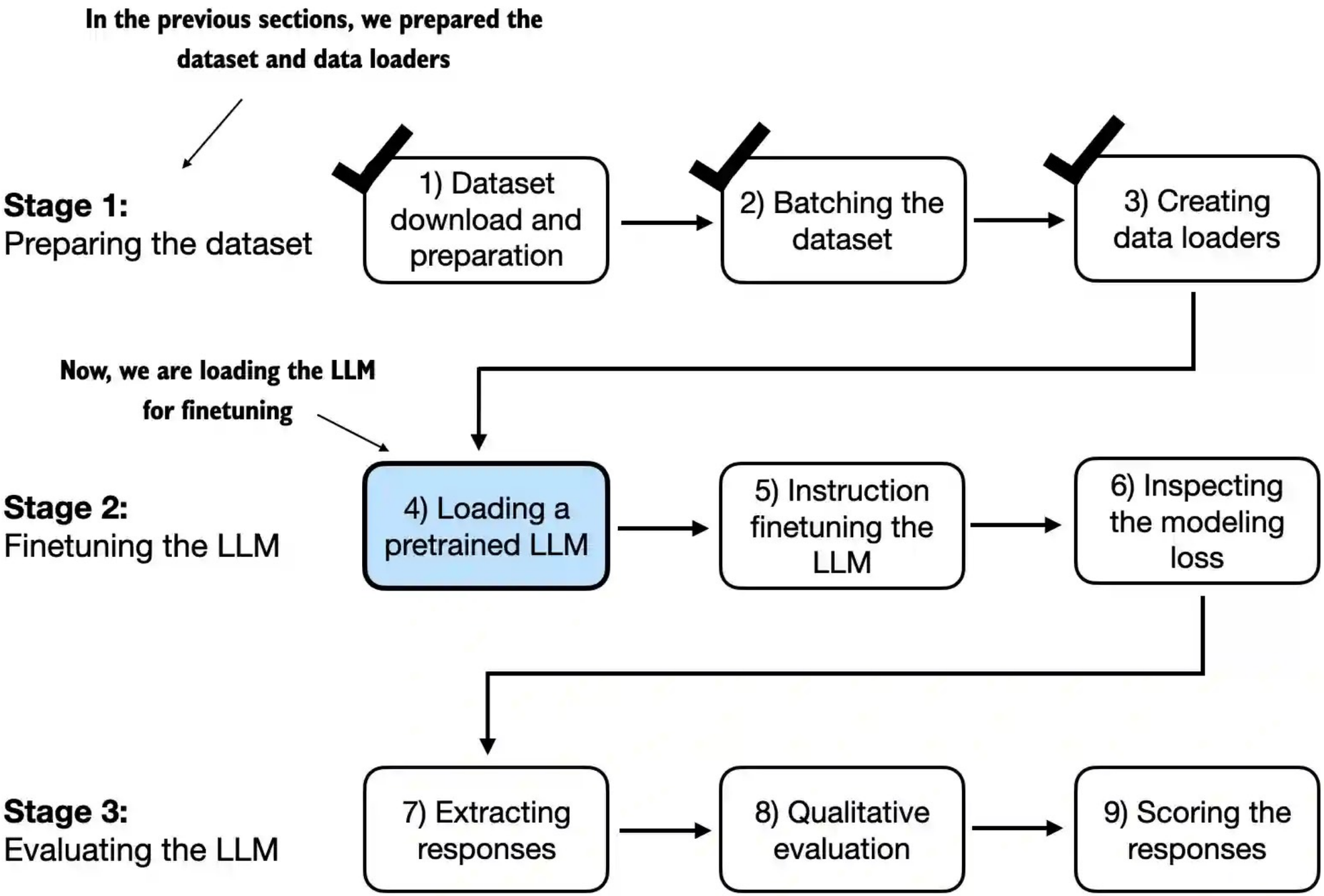
Dataset & Loader

```
rom torch.utils.data import DataLoader

num_workers = 0
batch_size = 8

torch.manual_seed(123)

train_dataset = InstructionDataset(train_data, tokenizer)
train_loader = DataLoader(
    train_dataset,
    batch_size=batch_size,
    collate_fn=customized_collate_fn,
    shuffle=True,
    drop_last=True,
    num_workers=num_workers
)
```


In the previous sections, we prepared the dataset and data loaders
Stage 1:
Preparing the dataset
1) Dataset download and preparation
2) Batching the dataset
3) Creating data loaders
Now, we are loading the LLM for finetuning
Stage 2:
Finetuning the LLM
4) Loading a pretrained LLM
5) Instruction finetuning the LLM
6) Inspecting the modeling loss
Stage 3:
Evaluating the LLM
7) Extracting responses
8) Qualitative evaluation
9) Scoring the responses

```
from gpt_download import download_and_load_gpt2
from previous_chapters import GPTModel, load_weights_into_gpt

BASE_CONFIG = {
    "vocab_size": 50257,     # Vocabulary size
    "context_length": 1024,  # Context length
    "drop_rate": 0.0,        # Dropout rate
    "qkv_bias": True         # Query-key-value bias
}

model_configs = {
    "gpt2-small (124M)": {"emb_dim": 768, "n_layers": 12, "n_heads": 12},
    "gpt2-medium (355M)": {"emb_dim": 1024, "n_layers": 24, "n_heads": 16},
    "gpt2-large (774M)": {"emb_dim": 1280, "n_layers": 36, "n_heads": 20},
    "gpt2-xl (1558M)": {"emb_dim": 1600, "n_layers": 48, "n_heads": 25},
}

CHOOSE_MODEL = "gpt2-medium (355M)"
BASE_CONFIG.update(model_configs[CHOOSE_MODEL])

model_size = CHOOSE_MODEL.split(" ")[-1].lstrip("(").rstrip(")")
settings, params = download_and_load_gpt2(
    model_size=model_size,
    models_dir="gpt2"
)

model = GPTModel(BASE_CONFIG)
load_weights_into_gpt(model, params)
model.eval();
```

```
def calc_loss_loader(data_loader, model, device, num_batches=None):
    total_loss = 0.
    if len(data_loader) == 0:
        return float("nan")
    elif num_batches is None:
        num_batches = len(data_loader)
    else:
        # Reduce the number of batches to match the total number of batches in the data loader
        # if num_batches exceeds the number of batches in the data loader
        num_batches = min(num_batches, len(data_loader))
    for i, (input_batch, target_batch) in enumerate(data_loader):
        if i < num_batches:
            loss = calc_loss_batch(input_batch, target_batch, model, device)
            total_loss += loss.item()
        else:
            break
    return total_loss / num_batches
```

```
def train_model_simple(model, train_loader, val_loader, optimizer, device, num_epochs,
                       eval_freq, eval_iter, start_context, tokenizer):
    train_losses, val_losses, track_tokens_seen = [], [], []
    tokens_seen, global_step = 0, -1

    # Main training loop
    for epoch in range(num_epochs):
        model.train()  # Set model to training mode

        for input_batch, target_batch in train_loader:
            optimizer.zero_grad()  # Reset loss gradients from previous batch iteration
            loss = calc_loss_batch(input_batch, target_batch, model, device)
            loss.backward()  # Calculate loss gradients
            optimizer.step()  # Update model weights using loss gradients
            tokens_seen += input_batch.numel()
            global_step += 1

            if global_step % eval_freq == 0:
                train_loss, val_loss = evaluate_model(
                    model, train_loader, val_loader, device, eval_iter)
                train_losses.append(train_loss)
                val_losses.append(val_loss)
                track_tokens_seen.append(tokens_seen)
                print(f"Ep {epoch+1} (Step {global_step:06d}): "
                      f"Train loss {train_loss:.3f}, Val loss {val_loss:.3f}")

        generate_and_print_sample(
            model, tokenizer, device, start_context
        )

    return train_losses, val_losses, track_tokens_seen
```

```
import time

start_time = time.time()

torch.manual_seed(123)

optimizer = torch.optim.AdamW(model.parameters(), lr=0.00005, weight_decay=0.1)

num_epochs = 2

train_losses, val_losses, tokens_seen = train_model_simple(
    model, train_loader, val_loader, optimizer, device,
    num_epochs=num_epochs, eval_freq=5, eval_iter=5,
    start_context=format_input(val_data[0]), tokenizer=tokenizer
)

end_time = time.time()
execution_time_minutes = (end_time - start_time) / 60
print(f"Training completed in {execution_time_minutes:.2f} minutes.")
```

Initial losses

Training loss: 3.8390236377716063

Validation loss: 3.761903762817383

Ep 1 (Step 000000): Train loss 2.637, Val loss 2.626

Ep 1 (Step 000005): Train loss 1.174, Val loss 1.102

Ep 1 (Step 000010): Train loss 0.872, Val loss 0.945

Ep 1 (Step 000015): Train loss 0.856, Val loss 0.906

Ep 1 (Step 000020): Train loss 0.776, Val loss 0.881

Ep 1 (Step 000025): Train loss 0.753, Val loss 0.859

Ep 1 (Step 000030): Train loss 0.798, Val loss 0.836

Ep 2 (Step 000220): Train loss 0.299, Val loss 0.650

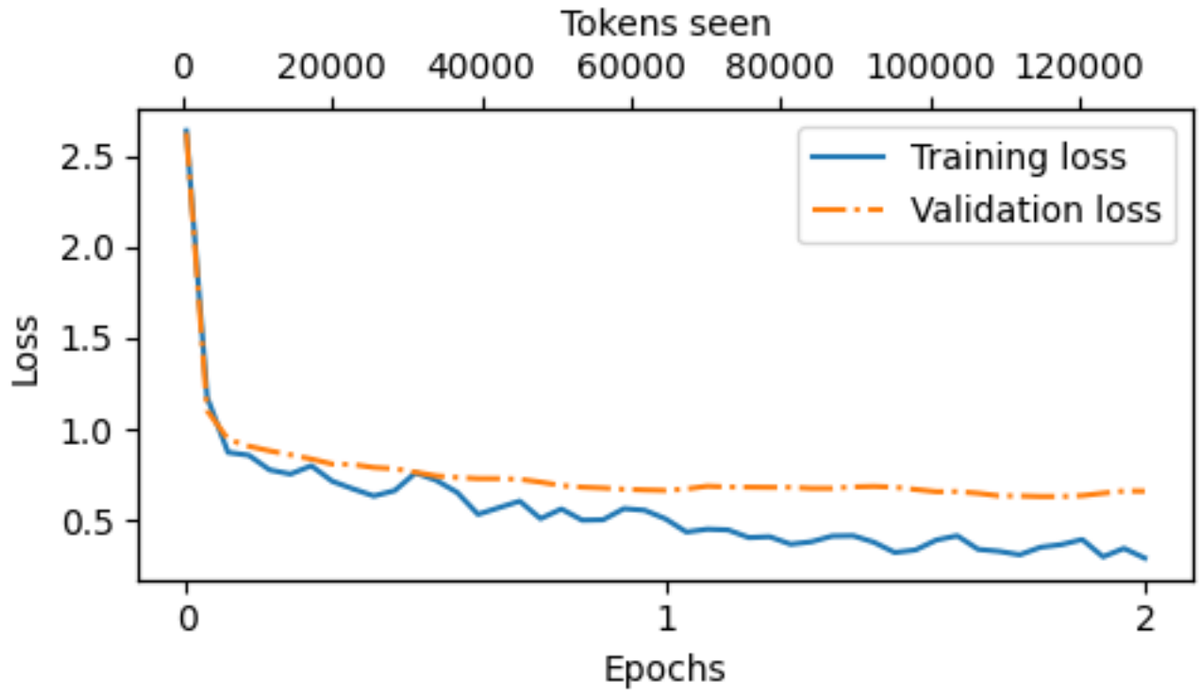
Ep 2 (Step 000225): Train loss 0.348, Val loss 0.662

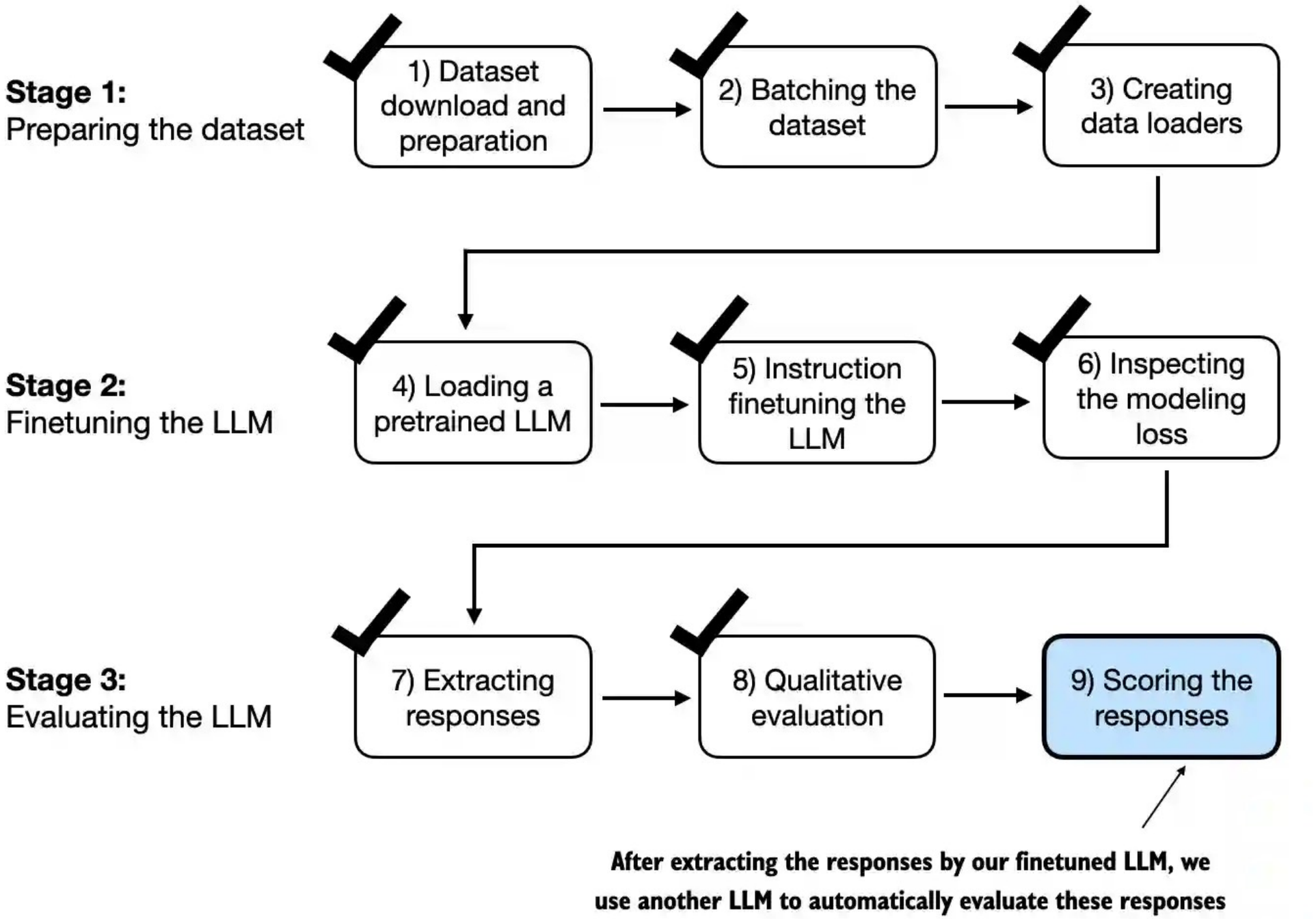
Ep 2 (Step 000230): Train loss 0.294, Val loss 0.657

Below is an instruction that describes a task. Write a response that appropriately completes the request. ### Instruction: Convert the active sentence to passive: 'The chef cooks the meal every day.' ### Response: The meal is cooked every day by the chef.<|endoftext|>The following is an instruction that describes a task. Write a response that appropriately completes the request. ### Instruction: What is the capital of the United Kingdom

Training completed in 22.82 minutes.

Plot saved as loss-plot-standalone.pdf


Tokens seen
0
20000
40000
60000
80000
100000
120000
Training loss
Validation loss
Loss
2.5
2.0
1.5
1.0
0.5
0
1
2
Epochs


Stage 1:
Preparing the dataset
1) Dataset download and preparation
2) Batching the dataset
3) Creating data loaders
Stage 2:
Finetuning the LLM
4) Loading a pretrained LLM
5) Instruction finetuning the LLM
6) Inspecting the modeling loss
Stage 3:
Evaluating the LLM
7) Extracting responses
8) Qualitative evaluation
9) Scoring the responses
After extracting the responses by our finetuned LLM, we use another LLM to automatically evaluate these responses

Save Model Responses

```
from tqdm import tqdm

for i, entry in tqdm(enumerate(test_data), total=len(test_data)):

    input_text = format_input(entry)

    token_ids = generate(
        model=model,
        idx=text_to_token_ids(input_text, tokenizer).to(device),
        max_new_tokens=256,
        context_size=BASE_CONFIG["context_length"],
        eos_id=50256
    )
    generated_text = token_ids_to_text(token_ids, tokenizer)
    response_text = generated_text[len(input_text):].replace("### Response:", "").strip()

    test_data[i]["model_response"] = response_text



with open("instruction-data-with-response.json", "w") as file:
    json.dump(test_data, file, indent=4)  # "indent" for pretty-printing
```

Sample Results

instruction"Correct the punctuation in the sentence."
input"Its time to go home."
output"The corrected sentence should be: 'It's time to go home.'"
model_response"The punctuation in the sentence is incorrect."

instruction"Classify the following items as either solid, liquid, or gas."
input"Mercury, oxygen, wood"
output"Mercury - Liquid Oxygen - Gas Wood - Solid"
model_response"Solid: Mercury Liquid: Oxygen Gas: Wood"

instruction"Edit the given text to ensure all plural nouns are spelled correctly."
input"The birds sings beautiful songs."
output"The birds sing beautiful songs."
model_response"The birds sings beautiful songs."

instruction"Generate a sentence using the word 'generous'."
input""
output"He is very generous and always helps those in need."
model_response"She was very generous and gave the poor man a meal."

Saving The Model

```
import re


file_name = f"{re.sub(r'[ ()]', '', CHOOSE_MODEL) }-sft.pth"
torch.save(model.state_dict(), file_name)
print(f"Model saved as {file_name}”)

# Load model via
# model.load_state_dict(torch.load(“gpt2-medium355M-sft.pth"))
```

Evaluation Techniques

Short-answer and multiple choice benchmarks

MMLU ("Measuring Massive Multitask Language Understanding", https://arxiv.org/abs/2009.03300)

Human preference comparison to other LLMs,

LMSYS chatbot arena (https://arena.lmsys.org)

Automated conversational benchmarks,

Another LLM evaluates the responses

AlpacaEval (https://tatsu-lab.github.io/alpaca_eval/)

Using Ollama to Evauate Model

```
import urllib.request

def query_model(
    prompt,
    model="llama3",
    url="http://127.0.0.1:11434/api/chat/"
):
    # Create the data payload as a dictionary
    data = {
    "model": model,
    "messages": [
        {"role": "user", "content": prompt}
    ],
    "options": {     # Settings below are required for deterministic responses
        "seed": 123,
        "temperature": 0,
        "num_ctx": 2048
    }
}
```

Using Ollama to Evauate Model

```
    # Convert the dictionary to a JSON formatted string and encode it to bytes
    payload = json.dumps(data).encode("utf-8")

    request = urllib.request.Request(
        url,
        data=payload,
        method="POST"
    )
    request.add_header("Content-Type", "application/json")

    # Send the request and capture the response
    response_data = ""
    with urllib.request.urlopen(request) as response:
        # Read and decode the response
        while True:
            line = response.readline().decode("utf-8")
            if not line:
                break
            response_json = json.loads(line)
            response_data += response_json["message"]["content"]

    return response_data

model = "llama3"
result = query_model("What do Llamas eat?", model)
print(result)
```

```
for entry in test_data[:3]:
    prompt = (
        f"Given the input `{format_input(entry)}` "
        f"and correct output `{entry['output']}`, "
        f"score the model response `{entry['model_response']}`"
        f" on a scale from 0 to 100, where 100 is the best score. "
    )
    print("\nDataset response:")
    print(">>", entry['output'])
    print("\nModel response:")
    print(">>", entry["model_response"])
    print("\nScore:")
    print(">>", query_model(prompt))
    print("\n-------------------------")
```

instruction"Rewrite the sentence using a simile."
input"The car is very fast."
output"The car is as fast as lightning."
model_response"The car is as fast as a bullet."

Dataset response:
>> The car is as fast as lightning.

Model response:
>> The car is as fast as a bullet.

Score:
>> I'd rate the model response "The car is as fast as a bullet." an 85 out of 100.

Here's why:

* The response uses a simile correctly, comparing the speed of the car to something else (in this case, a bullet).
* The comparison is relevant and makes sense, as bullets are known for their high velocity.
* The phrase "as fast as" is used correctly to introduce the simile.

The only reason I wouldn't give it a perfect score is that some people might find the comparison slightly less vivid or evocative than others. For example, comparing something to lightning (as in the original response) can be more dramatic and attention-grabbing. However, "as fast as a bullet" is still a strong and effective simile that effectively conveys the idea of the car's speed.

Overall, I think the model did a great job!

The Test

```
def generate_model_scores(json_data, json_key, model="llama3"):
    scores = []
    for entry in tqdm(json_data, desc="Scoring entries"):
        prompt = (
            f"Given the input `{format_input(entry)}` "
            f"and correct output `{entry['output']}`, "
            f"score the model response `{entry[json_key]}`"
            f" on a scale from 0 to 100, where 100 is the best score. "
            f"Respond with the integer number only."
        )
        score = query_model(prompt, model)
        try:
            scores.append(int(score))
        except ValueError:
            print(f"Could not convert score: {score}")
            continue

    return scores

scores = generate_model_scores(test_data, "model_response")
print(f"Number of scores: {len(scores)} of {len(test_data)}")
print(f"Average score: {sum(scores)/len(scores):.2f}\n")
```

```
Number of scores: 110 of 110
Average score: 50.32
```

```
Number of scores: 110 of 110
Average score: 47.63
```

```
Llama 3 8B base model score: 58.51
Llama 3 8B instruct model score: 82.65
```