

Group Project LLM

- $r=2,4,8,16$, epoch=10
- seed=42
- evaluation:
 - accuracy, f1, precision, recall
 - efficiency (time, trainable parameters, trainable parameters ratio, convergence)

```
In [36]: import warnings
warnings.filterwarnings("default", module="__main__")
warnings.filterwarnings("ignore", module=".*")
```

Base Model: DistilBERT

```
In [2]: # ===== BASELINE DISTILBERT =====

import os, time, random
import numpy as np
import torch

from datasets import load_dataset
from transformers import (
    DistilBertTokenizerFast,
    DistilBertForSequenceClassification,
    DataCollatorWithPadding,
    TrainingArguments,
    Trainer,
    set_seed
)
import evaluate

SEED = 42
DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
DATASET = "stanfordnlp/sst2"
TEXT_COL = "sentence"
LABEL_COL = "label"
NUM_EPOCHS = 10
BATCH_SIZE = 16
LR = 2e-5

'''
for IMDB dataset:
DATASET = "imdb"
TEXT_COL = "text"
LABEL_COL = "label"
'''

def set_all_seeds(seed=42):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)
    set_seed(seed)

set_all_seeds(SEED)

# ----- Load dataset and split (8:1:1) -----
raw = load_dataset(DATASET)
train_full = raw["train"]

train_temp = train_full.train_test_split(test_size=0.2, seed=SEED)
train_ds = train_temp["train"]
temp = train_temp["test"]

val_test = temp.train_test_split(test_size=0.5, seed=SEED)
```

```

val_ds = val_test["train"]
test_ds = val_test["test"]

# ----- Tokenization -----
tokenizer = DistilBertTokenizerFast.from_pretrained("distilbert-base-uncased")
def preprocess(x):
    return tokenizer(x[TEXT_COL], truncation=True, max_length=128)

train_ds = train_ds.map(preprocess, batched=True)
val_ds = val_ds.map(preprocess, batched=True)
test_ds = test_ds.map(preprocess, batched=True)

train_ds = train_ds.rename_column(LABEL_COL, "labels")
val_ds = val_ds.rename_column(LABEL_COL, "labels")
test_ds = test_ds.rename_column(LABEL_COL, "labels")

cols = ["input_ids", "attention_mask", "labels"]
train_ds.set_format(type="torch", columns=cols)
val_ds.set_format(type="torch", columns=cols)
test_ds.set_format(type="torch", columns=cols)

collator = DataCollatorWithPadding(tokenizer)

# ----- Metrics -----
acc = evaluate.load("accuracy")
f1 = evaluate.load("f1")
prec = evaluate.load("precision")
rec = evaluate.load("recall")

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=-1)
    return {
        "accuracy": acc.compute(predictions=preds, references=labels)["accuracy"],
        "f1": f1.compute(predictions=preds, references=labels, average="binary")["f1"],
        "precision": prec.compute(predictions=preds, references=labels, average="binary")["precision"],
        "recall": rec.compute(predictions=preds, references=labels, average="binary")["recall"],
    }

# ----- Model -----
model = DistilBertForSequenceClassification.from_pretrained(
    "distilbert-base-uncased", num_labels=2
).to(DEVICE)

total_params = sum(p.numel() for p in model.parameters())
trainable_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
ratio = trainable_params / total_params

print(f"Baseline: total={total_params}, trainable={trainable_params}, ratio={ratio:.4%}")

# ----- Train -----
args = TrainingArguments(
    output_dir="./baseline_distilbert",
    num_train_epochs=10,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    learning_rate=2e-5,
    eval_strategy="epoch",
    save_strategy="epoch",
    logging_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="f1",
    greater_is_better=True,
    seed=SEED,
    report_to="none",
)

trainer = Trainer(
    model=model,
    args=args,
    train_dataset=train_ds,

```

```

eval_dataset=val_ds,
data_collator=collator,
tokenizer=tokenizer,
compute_metrics=compute_metrics,
)

start = time.time()
trainer.train()
end = time.time()

print(f"Baseline training time: {end-start:.2f}s")
print("Eval:", trainer.evaluate(test_ds))
print("Convergence history:", trainer.state.log_history)

```

Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight', 'pre_classifier.bias', 'pre_classifier.weight']

You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

/tmp/ipykernel_2836370/2350084452.py:110: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class` instead.

```
trainer = Trainer(
```

Baseline: total=66955010, trainable=66955010, ratio=100.0000%

/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.

```
warnings.warn(
```

[16840/16840 27:00, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.238700	0.175075	0.941203	0.945409	0.943848	0.946976
2	0.125100	0.178515	0.944024	0.947704	0.952062	0.943386
3	0.088600	0.206909	0.944172	0.948521	0.940538	0.956642
4	0.061500	0.221665	0.944469	0.948385	0.947862	0.948909
5	0.047500	0.239832	0.947587	0.950979	0.956425	0.945595
6	0.033500	0.305291	0.945805	0.949759	0.946762	0.952775
7	0.027500	0.310512	0.946102	0.949966	0.948266	0.951671
8	0.018200	0.368410	0.946102	0.949924	0.949008	0.950842
9	0.012600	0.406172	0.946845	0.950580	0.950317	0.950842
10	0.010900	0.424868	0.946993	0.950670	0.951327	0.950014


```

Eval: {'eval_loss': 0.23587322235107422, 'eval_accuracy': 0.9452115812917594, 'eval_f1': 0.9513513513513514,
'eval_precision': 0.9504741833508957, 'eval_recall': 0.952230139878596, 'eval_runtime': 8.7337, 'eval_samples_per_second': 771.149, 'eval_steps_per_second': 24.159, 'epoch': 10.0}
Convergence history: [{'loss': 0.2387, 'grad_norm': 10.51264762878418, 'learning_rate': 1.800118764845606e-05, 'epoch': 1.0, 'step': 1684}, {'eval_loss': 0.17507508397102356, 'eval_accuracy': 0.9412026726057906, 'eval_f1': 0.9454094292803971, 'eval_precision': 0.9438480594549958, 'eval_recall': 0.9469759734879868, 'eval_runtime': 9.6709, 'eval_samples_per_second': 696.421, 'eval_steps_per_second': 21.818, 'epoch': 1.0, 'step': 1684}, {'loss': 0.1251, 'grad_norm': 5.240935325622559, 'learning_rate': 1.6001187648456057e-05, 'epoch': 2.0, 'step': 3368}, {'eval_loss': 0.1785149723291397, 'eval_accuracy': 0.9440237564959169, 'eval_f1': 0.9477042585656817, 'eval_precision': 0.9520624303232998, 'eval_recall': 0.9433858050262358, 'eval_runtime': 9.4749, 'eval_samples_per_second': 710.827, 'eval_steps_per_second': 22.269, 'epoch': 2.0, 'step': 3368}, {'loss': 0.0886, 'grad_norm': 11.543757438659668, 'learning_rate': 1.4001187648456058e-05, 'epoch': 3.0, 'step': 5052}, {'eval_loss': 0.2069089114665985, 'eval_accuracy': 0.9441722345953972, 'eval_f1': 0.9485213581599123, 'eval_precision': 0.9405376052131414, 'eval_recall': 0.9566418116542391, 'eval_runtime': 9.5145, 'eval_samples_per_second': 707.863, 'eval_steps_per_second': 22.177, 'epoch': 3.0, 'step': 5052}, {'loss': 0.0615, 'grad_norm': 0.22485654056072235, 'learning_rate': 1.2001187648456058e-05, 'epoch': 4.0, 'step': 6736}, {'eval_loss': 0.22166526317596436, 'eval_accuracy': 0.9444691907943579, 'eval_f1': 0.948385316036434, 'eval_precision': 0.9478620689655173, 'eval_recall': 0.9489091411212373, 'eval_runtime': 9.9979, 'eval_samples_per_second': 673.64, 'eval_steps_per_second': 21.104, 'epoch': 4.0, 'step': 6736}, {'loss': 0.0475, 'grad_norm': 0.09435376524925232, 'learning_rate': 1.0001187648456059e-05, 'epoch': 5.0, 'step': 8420}, {'eval_loss': 0.2398320436477661, 'eval_accuracy': 0.9475872308834447, 'eval_f1': 0.950979030690182, 'eval_precision': 0.9564245810055866, 'eval_recall': 0.9455951394642363, 'eval_runtime': 9.6024, 'eval_samples_per_second': 701.39, 'eval_steps_per_second': 21.974, 'epoch': 5.0, 'step': 8420}, {'loss': 0.0335, 'grad_norm': 0.45556309819221497, 'learning_rate': 8.001187648456058e-06, 'epoch': 6.0, 'step': 10104}, {'eval_loss': 0.3052908778190613, 'eval_accuracy': 0.9458054936896808, 'eval_f1': 0.9497591190640055, 'eval_precision': 0.9467618002195389, 'eval_recall': 0.9527754763877382, 'eval_runtime': 9.7726, 'eval_samples_per_second': 689.169, 'eval_steps_per_second': 21.591, 'epoch': 6.0, 'step': 10104}, {'loss': 0.0275, 'grad_norm': 8.011517524719238, 'learning_rate': 6.001187648456057e-06, 'epoch': 7.0, 'step': 11788}, {'eval_loss': 0.3105123043060303, 'eval_accuracy': 0.9461024498886415, 'eval_f1': 0.9499655410062027, 'eval_precision': 0.9482663731425427, 'eval_recall': 0.9516708091687379, 'eval_runtime': 9.3925, 'eval_samples_per_second': 717.063, 'eval_steps_per_second': 22.465, 'epoch': 7.0, 'step': 11788}, {'loss': 0.0182, 'grad_norm': 6.816328525543213, 'learning_rate': 4.001187648456058e-06, 'epoch': 8.0, 'step': 13472}, {'eval_loss': 0.36840957403182983, 'eval_accuracy': 0.9461024498886415, 'eval_f1': 0.9499241274658573, 'eval_precision': 0.9490077177508269, 'eval_recall': 0.9508423087544877, 'eval_runtime': 9.4225, 'eval_samples_per_second': 714.781, 'eval_steps_per_second': 22.393, 'epoch': 8.0, 'step': 13472}, {'loss': 0.0126, 'grad_norm': 2.9003257751464844, 'learning_rate': 2.001187648456057e-06, 'epoch': 9.0, 'step': 15156}, {'eval_loss': 0.4061717391014099, 'eval_accuracy': 0.9468448403860431, 'eval_f1': 0.9505797901711761, 'eval_precision': 0.9503174165056583, 'eval_recall': 0.9508423087544877, 'eval_runtime': 9.7566, 'eval_samples_per_second': 690.301, 'eval_steps_per_second': 21.626, 'epoch': 9.0, 'step': 15156}, {'loss': 0.0109, 'grad_norm': 0.011784575879573822, 'learning_rate': 1.1876484560570071e-09, 'epoch': 10.0, 'step': 16840}, {'eval_loss': 0.23587322235107422, 'eval_accuracy': 0.9452115812917594, 'eval_f1': 0.9513513513513514, 'eval_precision': 0.9504741833508957, 'eval_recall': 0.952230139878596, 'eval_runtime': 8.7337, 'eval_samples_per_second': 771.149, 'eval_steps_per_second': 24.159, 'epoch': 10.0, 'step': 16840}]

```

```

In [10]: import pandas as pd
# ----- Final Evaluation -----
final_metrics = trainer.evaluate(test_ds)

# ----- Save metrics -----
os.makedirs("./baseline_distilbert", exist_ok=True)
with open("./baseline_distilbert/final_metrics.json", "w") as f:
    json.dump(final_metrics, f, indent=4)

print("Saved final metrics to baseline_distilbert/final_metrics.json")

# ----- Save model -----
trainer.save_model("./baseline_distilbert/final_model")
print("Saved model to baseline_distilbert/final_model")

# ----- Convergence history -----
log_history = trainer.state.log_history
df_logs = pd.DataFrame(trainer.state.log_history)
# Separate clean tables
df_train = df_logs[df_logs["loss"].notnull()].reset_index(drop=True)
df_eval = df_logs[df_logs["eval_loss"].notnull()].reset_index(drop=True)

df_train.to_csv("./baseline_distilbert/train_log.csv", index=False)
df_eval.to_csv("./baseline_distilbert/eval_log.csv", index=False)

```

/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.

warnings.warn(

Saved final metrics to baseline_distilbert/final_metrics.json

Saved model to baseline_distilbert/final_model

In [28]: `import matplotlib.pyplot as plt`

```
df_eval_clean = df_eval.groupby("epoch").last().reset_index()
```

```
plt.figure(figsize=(8,5))
```

```
plt.plot(df_train["epoch"], df_train["loss"], label="Train Loss")
```

```
plt.plot(df_eval_clean["epoch"], df_eval_clean["eval_loss"], label="Eval Loss")
```

```
plt.xlabel("Epoch")
```

```
plt.ylabel("Loss")
```

```
plt.title("Convergence: Train vs Eval Loss")
```

```
plt.legend()
```

```
plt.grid()
```

```
plt.savefig("./baseline_distilbert/loss_curve.png", dpi=300)
```

```
plt.show()
```

```
plt.figure(figsize=(8,5))
```

```
plt.plot(df_eval_clean["epoch"], df_eval_clean["eval_accuracy"], label="Eval Accuracy")
```

```
plt.plot(df_eval_clean["epoch"], df_eval_clean["eval_f1"], label="Eval F1")
```

```
plt.xlabel("Epoch")
```

```
plt.ylabel("Metric")
```

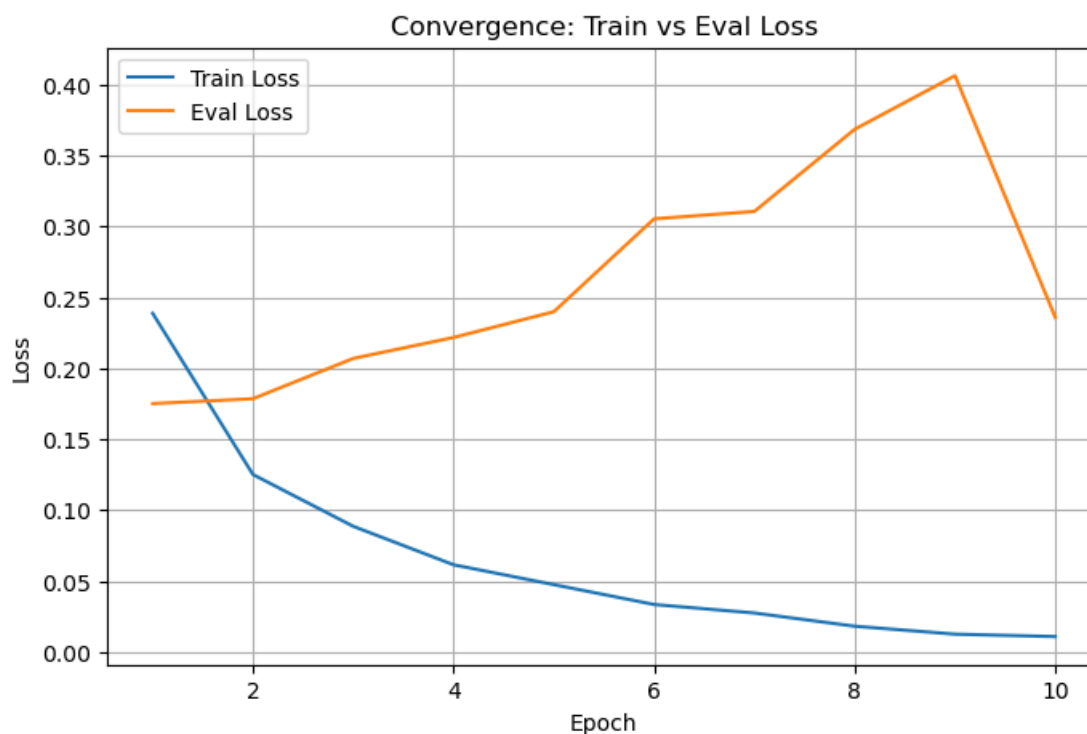
```
plt.title("Evaluation: Accuracy & F1")
```

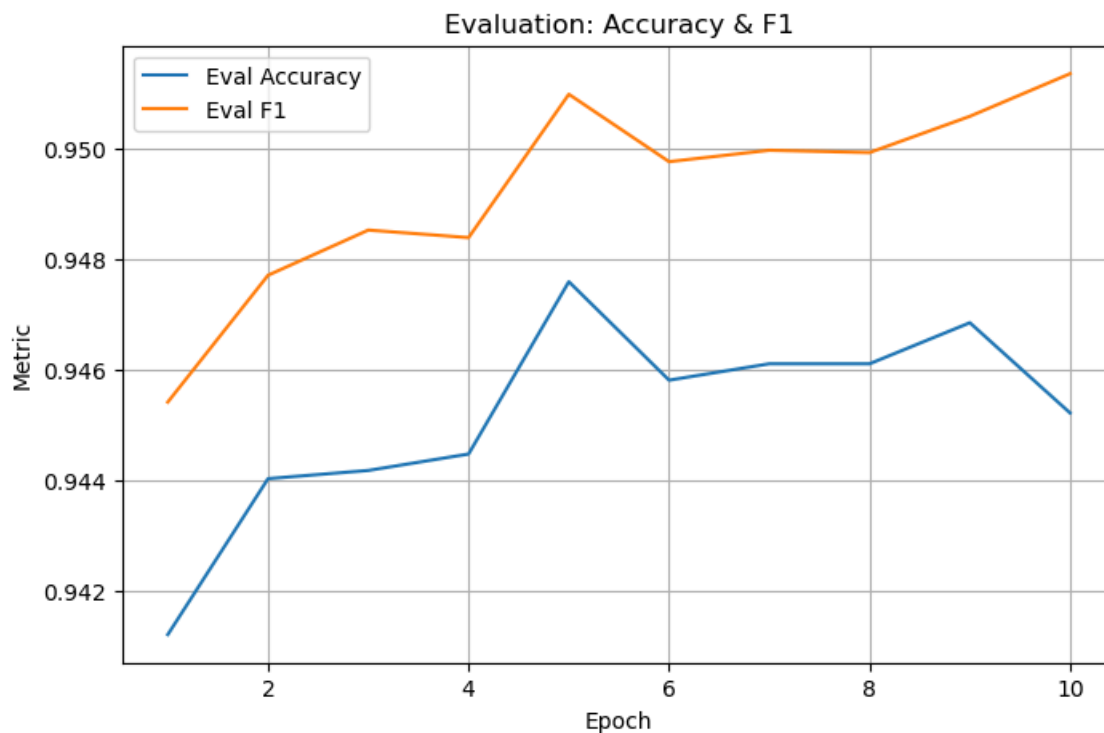
```
plt.legend()
```

```
plt.grid()
```

```
plt.savefig("./baseline_distilbert/metric_curve.png", dpi=300)
```

```
plt.show()
```





Sparse LoRA

```
In [34]: # ===== SPARSE LoRA MODEL =====

from typing import Dict, Any, List, Optional
import math
from peft import LoraConfig, get_peft_model

# ----- Sparse LoRA config -----
RANKS: List[int] = [2, 4, 8, 16]
L1_LAMBDA = 1e-5 # sparsity strength for LoRA weights

def count_trainable_params(model: torch.nn.Module) -> int:
    return sum(p.numel() for p in model.parameters() if p.requires_grad)

def count_total_params(model: torch.nn.Module) -> int:
    return sum(p.numel() for p in model.parameters())

def compute_lora_sparsity(model: torch.nn.Module, threshold: float = 1e-3) -> float:
    """
    Approximate sparsity: fraction of LoRA parameters with |w| < threshold.
    """
    total = 0
    near_zero = 0
    for name, param in model.named_parameters():
        if "lora_" in name and param.requires_grad:
            data = param.detach().abs()
            total += data.numel()
            near_zero += (data < threshold).sum().item()
    return near_zero / total if total > 0 else math.nan

class SparseLoraTrainer(Trainer):
    """
    Trainer with L1 penalty only on LoRA parameters.
    """
    def __init__(self, l1_lambda: float = 0.0, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.l1_lambda = l1_lambda
```

```

def compute_loss(
    self,
    model,
    inputs,
    return_outputs: bool = False,
    num_items_in_batch: Optional[int] = None, # NEW ARG
):
    # standard HF behavior
    outputs = model(*inputs)
    loss = outputs.loss

    # add L1 penalty on LoRA weights
    if self.l1_lambda > 0:
        l1_reg = 0.0
        for name, param in model.named_parameters():
            if "lora_" in name and param.requires_grad:
                l1_reg = l1_reg + param.abs().sum()
        loss = loss + self.l1_lambda * l1_reg

    return (loss, outputs) if return_outputs else loss

```

```

results_per_rank: List[Dict[str, Any]] = []

```

```

for r in RANKS:
    print("\n" + "=" * 80)
    print(f"Training Sparse LoRA DistilBERT with rank = {r}, epochs = {NUM_EPOCHS}")
    print("=" * 80)

    set_all_seeds(SEED)

    # Base DistilBERT for this rank
    base_model = DistilBertForSequenceClassification.from_pretrained(
        "distilbert-base-uncased",
        num_labels=2,
    )

    # LoRA config: attention projections in DistilBERT
    lora_config = LoraConfig(
        r=r,
        lora_alpha=2 * r,
        lora_dropout=0.1,
        bias="none",
        task_type="SEQ_CLS", # sequence classification
        target_modules=["q_lin", "k_lin", "v_lin", "out_lin"],
    )

    lora_model = get_peft_model(base_model, lora_config)
    lora_model.to(DEVICE)

    total_params = count_total_params(lora_model)
    trainable_params = count_trainable_params(lora_model)
    param_ratio = trainable_params / total_params

    print(f"[Rank {r}] total params: {total_params:,}")
    print(f"[Rank {r}] trainable params: {trainable_params:,}")
    print(f"[Rank {r}] trainable params ratio (trainable / total): {param_ratio:.4%}")

    training_args_lora = TrainingArguments(
        output_dir=f"./sparse_lora_rank{r}",
        num_train_epochs=NUM_EPOCHS,
        per_device_train_batch_size=BATCH_SIZE,
        per_device_eval_batch_size=BATCH_SIZE,
        learning_rate=LR,
        weight_decay=0.01,
        eval_strategy="epoch",
        save_strategy="epoch",
        logging_strategy="epoch",
        load_best_model_at_end=True,
        metric_for_best_model="f1",
        greater_is_better=True,
    )

```



```

        seed=SEED,
        report_to="none",
    )

    trainer = SparseLoraTrainer(
        l1_lambda=L1_LAMBDA,
        model=lora_model,
        args=training_args_lora,
        train_dataset=train_ds,
        eval_dataset=val_ds,
        tokenizer=tokenizer,
        data_collator=collator,
        compute_metrics=compute_metrics,
    )

    start_time = time.time()
    train_output = trainer.train()
    end_time = time.time()
    train_time = end_time - start_time
    print(f"[Rank {r}] Training time: {train_time:.2f} seconds")

    val_metrics = trainer.evaluate(eval_dataset=val_ds)
    test_metrics = trainer.evaluate(eval_dataset=test_ds)

    lora_sparsity = compute_lora_sparsity(lora_model, threshold=1e-3)
    print(f"[Rank {r}] Validation metrics: {val_metrics}")
    print(f"[Rank {r}] Test metrics: {test_metrics}")
    print(f"[Rank {r}] LoRA sparsity (<1e-3): {lora_sparsity:.2%}")
    print(f"[Rank {r}] Convergence log_history:")
    print(trainer.state.log_history)

    results_per_rank.append(
        {
            "rank": r,
            "total_params": total_params,
            "trainable_params": trainable_params,
            "param_ratio": param_ratio,
            "train_time_sec": train_time,
            "val_metrics": val_metrics,
            "test_metrics": test_metrics,
            "lora_sparsity(<1e-3)": lora_sparsity,
            "log_history": trainer.state.log_history, # convergence
        }
    )

print("\n\n=== Summary over ranks (Sparse LoRA) ===")
for res in results_per_rank:
    r = res["rank"]
    print(f"\nRank {r}:")
    print(f"  Params: {res['trainable_params']:,} / {res['total_params']:,} "
          f"({res['param_ratio']:.2%}")
    print(f"  Train time: {res['train_time_sec']:.2f} s")
    print(f"  Val F1: {res['val_metrics'].get('eval_f1', float('nan')):.4f}, "
          f"  f"Acc: {res['val_metrics'].get('eval_accuracy', float('nan')):.4f}")
    print(f"  Test F1: {res['test_metrics'].get('eval_f1', float('nan')):.4f}, "
          f"  f"Acc: {res['test_metrics'].get('eval_accuracy', float('nan')):.4f}")
    print(f"  LoRA sparsity (<1e-3): {res['lora_sparsity(<1e-3)']:.2%}")

```

```

=====
Training Sparse LoRA DistilBERT with rank = 2, epochs = 10
=====

```

Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight', 'pre_classifier.bias', 'pre_classifier.weight']

You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

[Rank 2] total params: 67,620,868

[Rank 2] trainable params: 665,858

[Rank 2] trainable params ratio (trainable / total): 0.9847%

/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.

warnings.warn(

[Rank 2] Validation metrics: {'eval_loss': 0.25018182396888733, 'eval_accuracy': 0.9003711952487008, 'eval_f1': 0.9072307479607356, 'eval_precision': 0.9083610188261351, 'eval_recall': 0.9061032863849765, 'eval_runtime': 10.8933, 'eval_samples_per_second': 618.271, 'eval_steps_per_second': 19.37, 'epoch': 10.0}

[Rank 2] Test metrics: {'eval_loss': 0.26539742946624756, 'eval_accuracy': 0.8914625092798812, 'eval_f1': 0.9028313172936329, 'eval_precision': 0.9094804499196572, 'eval_recall': 0.8962787015043547, 'eval_runtime': 11.0689, 'eval_samples_per_second': 608.46, 'eval_steps_per_second': 19.062, 'epoch': 10.0}

[Rank 2] LoRA sparsity (<1e-3): 15.55%

[Rank 2] Convergence log_history:

{'loss': 0.3939, 'grad_norm': 3.0737977027893066, 'learning_rate': 1.800118764845606e-05, 'epoch': 1.0, 'step': 1684}, {'eval_loss': 0.2961207926273346, 'eval_accuracy': 0.8764662212323683, 'eval_f1': 0.8823196605374823, 'eval_precision': 0.9043200927805161, 'eval_recall': 0.8613642640154653, 'eval_runtime': 11.1684, 'eval_samples_per_second': 603.042, 'eval_steps_per_second': 18.893, 'epoch': 1.0, 'step': 1684}, {'loss': 0.316, 'grad_norm': 1.479843258857727, 'learning_rate': 1.6001187648456057e-05, 'epoch': 2.0, 'step': 3368}, {'eval_loss': 0.28003761172294617, 'eval_accuracy': 0.8853749072011878, 'eval_f1': 0.8928670552317514, 'eval_precision': 0.897350069735007, 'eval_recall': 0.8884286108809721, 'eval_runtime': 11.0601, 'eval_samples_per_second': 608.944, 'eval_steps_per_second': 19.078, 'epoch': 2.0, 'step': 3368}, {'loss': 0.3028, 'grad_norm': 1.9990397691726685, 'learning_rate': 1.4001187648456058e-05, 'epoch': 3.0, 'step': 5052}, {'eval_loss': 0.27078408002853394, 'eval_accuracy': 0.8913140311804009, 'eval_f1': 0.8989508558807289, 'eval_precision': 0.8987027325420922, 'eval_recall': 0.8991991162662248, 'eval_runtime': 11.7675, 'eval_samples_per_second': 572.341, 'eval_steps_per_second': 17.931, 'epoch': 3.0, 'step': 5052}, {'loss': 0.2924, 'grad_norm': 3.801942825317383, 'learning_rate': 1.2001187648456058e-05, 'epoch': 4.0, 'step': 6736}, {'eval_loss': 0.26297202706336975, 'eval_accuracy': 0.8944320712694878, 'eval_f1': 0.9009887202339507, 'eval_precision': 0.9087078651685393, 'eval_recall': 0.8933996133664733, 'eval_runtime': 12.6223, 'eval_samples_per_second': 533.581, 'eval_steps_per_second': 16.716, 'epoch': 4.0, 'step': 6736}, {'loss': 0.2859, 'grad_norm': 1.7913497686386108, 'learning_rate': 1.0001187648456059e-05, 'epoch': 5.0, 'step': 8420}, {'eval_loss': 0.26198816299438477, 'eval_accuracy': 0.8966592427616926, 'eval_f1': 0.9044218621257896, 'eval_precision': 0.8994810161158153, 'eval_recall': 0.9094172880419773, 'eval_runtime': 12.471, 'eval_samples_per_second': 540.054, 'eval_steps_per_second': 16.919, 'epoch': 5.0, 'step': 8420}, {'loss': 0.2815, 'grad_norm': 1.631085991859436, 'learning_rate': 8.001187648456058e-06, 'epoch': 6.0, 'step': 10104}, {'eval_loss': 0.25459036231040955, 'eval_accuracy': 0.8978470675575353, 'eval_f1': 0.9042049568365358, 'eval_precision': 0.9118225217635496, 'eval_recall': 0.8967136150234741, 'eval_runtime': 11.6626, 'eval_samples_per_second': 577.485, 'eval_steps_per_second': 18.092, 'epoch': 6.0, 'step': 10104}, {'loss': 0.2779, 'grad_norm': 2.9907469749450684, 'learning_rate': 6.001187648456057e-06, 'epoch': 7.0, 'step': 11788}, {'eval_loss': 0.2526792883872986, 'eval_accuracy': 0.8991833704528582, 'eval_f1': 0.9049951028403526, 'eval_precision': 0.9171866137266024, 'eval_recall': 0.8931234465617233, 'eval_runtime': 11.6393, 'eval_samples_per_second': 578.645, 'eval_steps_per_second': 18.128, 'epoch': 7.0, 'step': 11788}, {'loss': 0.2759, 'grad_norm': 3.077967643737793, 'learning_rate': 4.001187648456058e-06, 'epoch': 8.0, 'step': 13472}, {'eval_loss': 0.2510381042957306, 'eval_accuracy': 0.8999257609502599, 'eval_f1': 0.9069574820541138, 'eval_precision': 0.9067071487717361, 'eval_recall': 0.9072079536039768, 'eval_runtime': 12.0695, 'eval_samples_per_second': 558.018, 'eval_steps_per_second': 17.482, 'epoch': 8.0, 'step': 13472}, {'loss': 0.2727, 'grad_norm': 1.2240530252456665, 'learning_rate': 2.001187648456057e-06, 'epoch': 9.0, 'step': 15156}, {'eval_loss': 0.25018182396888733, 'eval_accuracy': 0.9003711952487008, 'eval_f1': 0.9072307479607356, 'eval_precision': 0.9083610188261351, 'eval_recall': 0.9061032863849765, 'eval_runtime': 12.2437, 'eval_samples_per_second': 550.079, 'eval_steps_per_second': 17.233, 'epoch': 9.0, 'step': 15156}, {'loss': 0.2716, 'grad_norm': 2.7022695541381836, 'learning_rate': 1.1876484560570071e-09, 'epoch': 10.0, 'step': 16840}, {'eval_loss': 0.24939800798892975, 'eval_accuracy': 0.9003711952487008, 'eval_f1': 0.906792610084734, 'eval_precision': 0.9122414756847401, 'eval_recall': 0.9014084507042254, 'eval_runtime': 11.4602, 'eval_samples_per_second': 587.685, 'eval_steps_per_second': 18.412, 'epoch': 10.0, 'step': 16840}, {'train_runtime': 2040.4051, 'train_samples_per_second': 264.06, 'train_steps_per_second': 8.253, 'total_flos': 5672878557754200.0, 'train_loss': 0.2970552820491114, 'epoch': 10.0, 'step': 16840}, {'eval_loss': 0.25018182396888733, 'eval_accuracy': 0.9003711952487008, 'eval_f1': 0.9072307479607356, 'eval_precision': 0.9083610188261351, 'eval_recall': 0.9061032863849765, 'eval_runtime': 10.8933, 'eval_samples_per_second': 618.271, 'eval_steps_per_second': 19.37, 'epoch': 10.0, 'step': 16840}, {'eval_loss': 0.26539742946624756, 'eval_accuracy': 0.8914625092798812, 'eval_f1': 0.9028313172936329, 'eval_precision': 0.9094804499196572, 'eval_recall': 0.8962787015043547, 'eval_runtime': 11.0689, 'eval_samples_per_second': 608.46, 'eval_steps_per_second': 19.062, 'epoch': 10.0, 'step': 16840}]

=====
Training Sparse LoRA DistilBERT with rank = 4, epochs = 10
=====

Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight', 'pre_classifier.bias', 'pre_classifier.weight']

You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

[Rank 4] total params: 67,694,596

[Rank 4] trainable params: 739,586

[Rank 4] trainable params ratio (trainable / total): 1.0925%


```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
```

```
warnings.warn(
```

```
[Rank 4] Validation metrics: {'eval_loss': 0.245857372879982, 'eval_accuracy': 0.9067557535263548, 'eval_f1': 0.9134509371554576, 'eval_precision': 0.9116918844566713, 'eval_recall': 0.9152167909417288, 'eval_runtime': 11.4858, 'eval_samples_per_second': 586.376, 'eval_steps_per_second': 18.371, 'epoch': 10.0}
```

```
[Rank 4] Test metrics: {'eval_loss': 0.2627865672111511, 'eval_accuracy': 0.895025983667409, 'eval_f1': 0.9063948100092678, 'eval_precision': 0.9094048884165781, 'eval_recall': 0.9034045922406968, 'eval_runtime': 11.0578, 'eval_samples_per_second': 609.073, 'eval_steps_per_second': 19.082, 'epoch': 10.0}
```

```
[Rank 4] LoRA sparsity (<1e-3): 21.52%
```

```
[Rank 4] Convergence log_history:
```

```
{'loss': 0.3863, 'grad_norm': 3.1279044151306152, 'learning_rate': 1.800118764845606e-05, 'epoch': 1.0, 'step': 1684}, {'eval_loss': 0.2941770851612091, 'eval_accuracy': 0.8797327394209354, 'eval_f1': 0.8859154929577465, 'eval_precision': 0.9039954009772924, 'eval_recall': 0.8685446009389671, 'eval_runtime': 11.3424, 'eval_samples_per_second': 593.791, 'eval_steps_per_second': 18.603, 'epoch': 1.0, 'step': 1684}, {'loss': 0.3125, 'grad_norm': 1.6540402173995972, 'learning_rate': 1.6001187648456057e-05, 'epoch': 2.0, 'step': 3368}, {'eval_loss': 0.27636703848838806, 'eval_accuracy': 0.8902746844840386, 'eval_f1': 0.8978011340063615, 'eval_precision': 0.8991689750692521, 'eval_recall': 0.8964374482187241, 'eval_runtime': 11.4461, 'eval_samples_per_second': 588.41, 'eval_steps_per_second': 18.434, 'epoch': 2.0, 'step': 3368}, {'loss': 0.2987, 'grad_norm': 1.5005635023117065, 'learning_rate': 1.4001187648456058e-05, 'epoch': 3.0, 'step': 5052}, {'eval_loss': 0.26759597659111023, 'eval_accuracy': 0.8948775055679288, 'eval_f1': 0.902398676592225, 'eval_precision': 0.9009083402146986, 'eval_recall': 0.903893951946976, 'eval_runtime': 11.6671, 'eval_samples_per_second': 577.264, 'eval_steps_per_second': 18.085, 'epoch': 3.0, 'step': 5052}, {'loss': 0.2876, 'grad_norm': 4.587261199951172, 'learning_rate': 1.2001187648456058e-05, 'epoch': 4.0, 'step': 6736}, {'eval_loss': 0.25898805260658264, 'eval_accuracy': 0.8999257609502599, 'eval_f1': 0.9057606263982103, 'eval_precision': 0.9173038799207024, 'eval_recall': 0.8945042805854736, 'eval_runtime': 11.796, 'eval_samples_per_second': 570.957, 'eval_steps_per_second': 17.887, 'epoch': 4.0, 'step': 6736}, {'loss': 0.2805, 'grad_norm': 1.8961390256881714, 'learning_rate': 1.0001187648456059e-05, 'epoch': 5.0, 'step': 8420}, {'eval_loss': 0.2582024335861206, 'eval_accuracy': 0.9020044543429844, 'eval_f1': 0.9095642641819677, 'eval_precision': 0.902638020125102, 'eval_recall': 0.9165976249654791, 'eval_runtime': 11.2974, 'eval_samples_per_second': 596.155, 'eval_steps_per_second': 18.677, 'epoch': 5.0, 'step': 8420}, {'loss': 0.2754, 'grad_norm': 1.8083791732788086, 'learning_rate': 8.001187648456058e-06, 'epoch': 6.0, 'step': 10104}, {'eval_loss': 0.24969258904457092, 'eval_accuracy': 0.904083147735709, 'eval_f1': 0.910102978012027, 'eval_precision': 0.9172510518934082, 'eval_recall': 0.9030654515327258, 'eval_runtime': 11.8729, 'eval_samples_per_second': 567.259, 'eval_steps_per_second': 17.772, 'epoch': 6.0, 'step': 10104}, {'loss': 0.2716, 'grad_norm': 3.006859302520752, 'learning_rate': 6.001187648456057e-06, 'epoch': 7.0, 'step': 11788}, {'eval_loss': 0.24755080044269562, 'eval_accuracy': 0.9058648849294729, 'eval_f1': 0.9114772409941357, 'eval_precision': 0.9217735103078226, 'eval_recall': 0.9014084507042254, 'eval_runtime': 11.7975, 'eval_samples_per_second': 570.885, 'eval_steps_per_second': 17.885, 'epoch': 7.0, 'step': 11788}, {'loss': 0.2688, 'grad_norm': 3.449216842651367, 'learning_rate': 4.001187648456058e-06, 'epoch': 8.0, 'step': 13472}, {'eval_loss': 0.245857372879982, 'eval_accuracy': 0.9067557535263548, 'eval_f1': 0.9134509371554576, 'eval_precision': 0.9116918844566713, 'eval_recall': 0.9152167909417288, 'eval_runtime': 11.4212, 'eval_samples_per_second': 589.692, 'eval_steps_per_second': 18.474, 'epoch': 8.0, 'step': 13472}, {'loss': 0.2658, 'grad_norm': 1.5881074666976929, 'learning_rate': 2.001187648456057e-06, 'epoch': 9.0, 'step': 15156}, {'eval_loss': 0.2449660748243332, 'eval_accuracy': 0.9066072754268746, 'eval_f1': 0.9130615065653075, 'eval_precision': 0.913945766463752, 'eval_recall': 0.9121789560894781, 'eval_runtime': 12.0478, 'eval_samples_per_second': 559.024, 'eval_steps_per_second': 17.514, 'epoch': 9.0, 'step': 15156}, {'loss': 0.2651, 'grad_norm': 2.6178510189056396, 'learning_rate': 1.1876484560570071e-09, 'epoch': 10.0, 'step': 16840}, {'eval_loss': 0.24408331513404846, 'eval_accuracy': 0.9070527097253155, 'eval_f1': 0.9131520532741398, 'eval_precision': 0.9174797881237803, 'eval_recall': 0.9088649544324772, 'eval_runtime': 11.513, 'eval_samples_per_second': 584.993, 'eval_steps_per_second': 18.327, 'epoch': 10.0, 'step': 16840}, {'train_runtime': 2030.95, 'train_samples_per_second': 265.29, 'train_steps_per_second': 8.292, 'total_flos': 5682430532563800.0, 'train_loss': 0.29122069986302607, 'epoch': 10.0, 'step': 16840}, {'eval_loss': 0.245857372879982, 'eval_accuracy': 0.9067557535263548, 'eval_f1': 0.9134509371554576, 'eval_precision': 0.9116918844566713, 'eval_recall': 0.9152167909417288, 'eval_runtime': 11.4858, 'eval_samples_per_second': 586.376, 'eval_steps_per_second': 18.371, 'epoch': 10.0, 'step': 16840}, {'eval_loss': 0.2627865672111511, 'eval_accuracy': 0.895025983667409, 'eval_f1': 0.9063948100092678, 'eval_precision': 0.9094048884165781, 'eval_recall': 0.9034045922406968, 'eval_runtime': 11.0578, 'eval_samples_per_second': 609.073, 'eval_steps_per_second': 19.082, 'epoch': 10.0, 'step': 16840}]
```

```
=====
Training Sparse LoRA DistilBERT with rank = 8, epochs = 10
=====
```

```
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight', 'pre_classifier.bias', 'pre_classifier.weight']
```

```
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
/tmp/ipykernel_2836370/731623691.py:39: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `SparseLoraTrainer.__init__`. Use `processing_class` instead.
```

```
super().__init__(*args, **kwargs)
```


[Rank 8] total params: 67,842,052
[Rank 8] trainable params: 887,042
[Rank 8] trainable params ratio (trainable / total): 1.3075%

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

[16840/16840 34:14, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.386200	0.295880	0.884781	0.891134	0.905617	0.877106
2	0.313800	0.277604	0.894878	0.902237	0.902237	0.902237
3	0.299300	0.268710	0.897847	0.905546	0.900355	0.910798
4	0.287800	0.259213	0.904826	0.910362	0.922096	0.898923
5	0.280200	0.259396	0.904380	0.912094	0.901754	0.922673
6	0.273700	0.249641	0.908389	0.914222	0.920493	0.908036
7	0.269600	0.247296	0.911359	0.916468	0.928815	0.904446
8	0.266700	0.245347	0.910171	0.916632	0.914741	0.918531
9	0.263500	0.244415	0.911210	0.917220	0.919512	0.914941
10	0.262700	0.243555	0.912101	0.917823	0.922690	0.913007

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

[Rank 8] Training time: 2055.44 seconds

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

[211/211 00:22]

/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.

warnings.warn()

[Rank 8] Validation metrics: {'eval_loss': 0.24355515837669373, 'eval_accuracy': 0.9121009651076466, 'eval_f1': 0.917823431426985, 'eval_precision': 0.9226904828356126, 'eval_recall': 0.9130074565037283, 'eval_runtime': 11.4322, 'eval_samples_per_second': 589.124, 'eval_steps_per_second': 18.457, 'epoch': 10.0}
[Rank 8] Test metrics: {'eval_loss': 0.25989142060279846, 'eval_accuracy': 0.9017074981440237, 'eval_f1': 0.9122015915119364, 'eval_precision': 0.9168221807517996, 'eval_recall': 0.9076273423066772, 'eval_runtime': 11.906, 'eval_samples_per_second': 565.682, 'eval_steps_per_second': 17.722, 'epoch': 10.0}
[Rank 8] LoRA sparsity (<1e-3): 27.88%
[Rank 8] Convergence log_history:
{
 'loss': 0.3862, 'grad_norm': 2.9998624324798584, 'learning_rate': 1.800118764845606e-05, 'epoch': 1.0, 'step': 1684},
 {'eval_loss': 0.2958800792694092, 'eval_accuracy': 0.8847809948032666, 'eval_f1': 0.8911335578002245, 'eval_precision': 0.9056173367550613, 'eval_recall': 0.8771057718862193, 'eval_runtime': 11.3446, 'eval_samples_per_second': 593.675, 'eval_steps_per_second': 18.599, 'epoch': 1.0, 'step': 1684},
 {'loss': 0.3138, 'grad_norm': 1.8658512830734253, 'learning_rate': 1.6001187648456057e-05, 'epoch': 2.0, 'step': 3368},
 {'eval_loss': 0.27760350704193115, 'eval_accuracy': 0.8948775055679288, 'eval_f1': 0.9022369511184756, 'eval_precision': 0.9022369511184756, 'eval_recall': 0.9022369511184756, 'eval_runtime': 11.4885, 'eval_samples_per_second': 586.24, 'eval_steps_per_second': 18.366, 'epoch': 2.0, 'step': 3368},
 {'loss': 0.2993, 'grad_norm': 1.4917348623275757, 'learning_rate': 1.4001187648456058e-05, 'epoch': 3.0, 'step': 5052},
 {'eval_loss': 0.268709659576416, 'eval_accuracy': 0.8978470675575353, 'eval_f1': 0.9055464030752334, 'eval_precision': 0.9003549003549004, 'eval_recall': 0.9107981220657277, 'eval_runtime': 12.9269, 'eval_samples_per_second': 521.007, 'eval_steps_per_second': 16.323, 'epoch': 3.0, 'step': 5052},
 {'loss': 0.2878, 'grad_norm': 4.908908843994141, 'learning_rate': 1.2001187648456058e-05, 'epoch': 4.0, 'step': 6736},
 {'eval_loss': 0.25921282172203064, 'eval_accuracy': 0.9048255382331106, 'eval_f1': 0.9103621871066984, 'eval_precision': 0.9220963172804533, 'eval_recall': 0.8989229494614748, 'eval_runtime': 11.5367, 'eval_samples_per_second': 583.789, 'eval_steps_per_second': 18.289, 'epoch': 4.0, 'step': 6736},
 {'loss': 0.2802, 'grad_norm': 2.5097811222076416, 'learning_rate': 1.0001187648456059e-05, 'epoch': 5.0, 'step': 8420},
 {'eval_loss': 0.25939592719078064, 'eval_accuracy': 0.9043801039346696, 'eval_f1': 0.9120939120939121, 'eval_precision': 0.9017543859649123, 'eval_recall': 0.9226732946699807, 'eval_runtime': 12.1457, 'eval_samples_per_second': 554.515, 'eval_steps_per_second': 17.372, 'epoch': 5.0, 'step': 8420},
 {'loss': 0.2737, 'grad_norm': 2.011146068572998, 'learning_rate': 8.001187648456058e-06, 'epoch': 6.0, 'step': 10104},
 {'eval_loss': 0.2496410608291626, 'eval_accuracy': 0.9083890126206384, 'eval_f1': 0.914222160437755, 'eval_precision': 0.9204927211646137, 'eval_recall': 0.908036454018227, 'eval_runtime': 11.4293, 'eval_samples_per_second': 589.274, 'eval_steps_per_second': 18.461, 'epoch': 6.0, 'step': 10104},
 {'loss': 0.2696, 'grad_norm': 3.028676748275757, 'learning_rate': 6.001187648456057e-06, 'epoch': 7.0, 'step': 11788},
 {'eval_loss': 0.24729610979557037, 'eval_accuracy': 0.9113585746102449, 'eval_f1': 0.916468448299986, 'eval_precision': 0.9288145207033466, 'eval_recall': 0.9044462855564761, 'eval_runtime': 12.4119, 'eval_samples_per_second': 542.626, 'eval_steps_per_second': 17.0, 'epoch': 7.0, 'step': 11788},
 {'loss': 0.2667, 'grad_norm': 3.6933021545410156, 'learning_rate': 4.001187648456058e-06, 'epoch': 8.0, 'step': 13472},
 {'eval_loss': 0.24534666538238525, 'eval_accuracy': 0.9101707498144024, 'eval_f1': 0.9166322171696293, 'eval_precision': 0.9147414741474147, 'eval_recall': 0.9185307925987296, 'eval_runtime': 11.6864, 'eval_samples_per_second': 576.312, 'eval_steps_per_second': 18.055, 'epoch': 8.0, 'step': 13472},
 {'loss': 0.2635, 'grad_norm': 1.674749732017517, 'learning_rate': 2.001187648456057e-06, 'epoch': 9.0, 'step': 15156},
 {'eval_loss': 0.24441498517990112, 'eval_accuracy': 0.9112100965107647, 'eval_f1': 0.917220376522702, 'eval_precision': 0.919511518179295, 'eval_recall': 0.9149406241369787, 'eval_runtime': 11.3226, 'eval_samples_per_second': 594.826, 'eval_steps_per_second': 18.635, 'epoch': 9.0, 'step': 15156},
 {'loss': 0.2627, 'grad_norm': 2.0971415042877197, 'learning_rate': 1.1876484560570071e-09, 'epoch': 10.0, 'step': 16840},
 {'eval_loss': 0.24355515837669373, 'eval_accuracy': 0.9121009651076466, 'eval_f1': 0.917823431426985, 'eval_precision': 0.9226904828356126, 'eval_recall': 0.9130074565037283, 'eval_runtime': 12.0692, 'eval_samples_per_second': 558.032, 'eval_steps_per_second': 17.483, 'epoch': 10.0, 'step': 16840},
 {'train_runtime': 2055.0752, 'train_samples_per_second': 262.175, 'train_steps_per_second': 8.194, 'total_flos': 5701534482183000.0, 'train_loss': 0.290355515649936, 'epoch': 10.0, 'step': 16840},
 {'eval_loss': 0.24355515837669373, 'eval_accuracy': 0.9121009651076466, 'eval_f1': 0.917823431426985, 'eval_precision': 0.9226904828356126, 'eval_recall': 0.9130074565037283, 'eval_runtime': 11.4322, 'eval_samples_per_second': 589.124, 'eval_steps_per_second': 18.457, 'epoch': 10.0, 'step': 16840},
 {'eval_loss': 0.25989142060279846, 'eval_accuracy': 0.9017074981440237, 'eval_f1': 0.9122015915119364, 'eval_precision': 0.9168221807517996, 'eval_recall': 0.9076273423066772, 'eval_runtime': 11.906, 'eval_samples_per_second': 565.682, 'eval_steps_per_second': 17.722, 'epoch': 10.0, 'step': 16840}]

Training Sparse LoRA DistilBERT with rank = 16, epochs = 10

Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight', 'pre_classifier.bias', 'pre_classifier.weight']

You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

/tmp/ipykernel_2836370/731623691.py:39: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `SparseLoraTrainer.__init__`. Use `processing_class` instead.

super().__init__(*args, **kwargs)

[Rank 16] total params: 68,136,964
[Rank 16] trainable params: 1,181,954
[Rank 16] trainable params ratio (trainable / total): 1.7347%

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

[16840/16840 33:56, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.397100	0.309513	0.889087	0.895363	0.908471	0.882629
2	0.324600	0.288913	0.897996	0.905228	0.904355	0.906103
3	0.307300	0.277806	0.905419	0.912368	0.908991	0.915769
4	0.294300	0.267853	0.910022	0.914864	0.931084	0.899199
5	0.284800	0.267210	0.910022	0.916872	0.910875	0.922949
6	0.277500	0.257248	0.913289	0.918731	0.925947	0.911627
7	0.272300	0.254617	0.915516	0.920319	0.933523	0.907484
8	0.268700	0.252295	0.915071	0.920885	0.922416	0.919359
9	0.265500	0.251517	0.917001	0.922393	0.927415	0.917426
10	0.263500	0.250823	0.916852	0.922092	0.929072	0.915217

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

[Rank 16] Training time: 2037.07 seconds

```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.
  warnings.warn(
```

[211/211 00:23]


```
/hpc/group/yizhanglab/yh151/miniconda3/envs/my-conda-env/lib/python3.9/site-packages/torch/nn/parallel/_functions.py:71: UserWarning: Was asked to gather along dimension 0, but all input tensors were scalars; will instead unsqueeze and return a vector.  
  warnings.warn(
```

[Rank 16] Validation metrics: {'eval_loss': 0.25151702761650085, 'eval_accuracy': 0.9170007423904974, 'eval_f1': 0.9223934471747883, 'eval_precision': 0.9274148520379676, 'eval_recall': 0.9174261253797293, 'eval_runtime': 11.2976, 'eval_samples_per_second': 596.147, 'eval_steps_per_second': 18.677, 'epoch': 10.0}

[Rank 16] Test metrics: {'eval_loss': 0.2650555968284607, 'eval_accuracy': 0.9097253155159614, 'eval_f1': 0.9195128408790045, 'eval_precision': 0.9224435590969455, 'eval_recall': 0.9166006861968857, 'eval_runtime': 12.1809, 'eval_samples_per_second': 552.916, 'eval_steps_per_second': 17.322, 'epoch': 10.0}

[Rank 16] LoRA sparsity (<1e-3): 35.68%

[Rank 16] Convergence log_history:

```
{'loss': 0.3971, 'grad_norm': 3.299694776535034, 'learning_rate': 1.800118764845606e-05, 'epoch': 1.0, 'step': 1684}, {'eval_loss': 0.3095133304595947, 'eval_accuracy': 0.889086859688196, 'eval_f1': 0.8953634962879955, 'eval_precision': 0.908470722001137, 'eval_recall': 0.8826291079812206, 'eval_runtime': 11.8989, 'eval_samples_per_second': 566.018, 'eval_steps_per_second': 17.733, 'epoch': 1.0, 'step': 1684}, {'loss': 0.3246, 'grad_norm': 2.1254234313964844, 'learning_rate': 1.6001187648456057e-05, 'epoch': 2.0, 'step': 3368}, {'eval_loss': 0.28891271352767944, 'eval_accuracy': 0.8979955456570156, 'eval_f1': 0.9052283073527383, 'eval_precision': 0.9043550165380375, 'eval_recall': 0.9061032863849765, 'eval_runtime': 11.686, 'eval_samples_per_second': 576.329, 'eval_steps_per_second': 18.056, 'epoch': 2.0, 'step': 3368}, {'loss': 0.3073, 'grad_norm': 1.5638595819473267, 'learning_rate': 1.4001187648456058e-05, 'epoch': 3.0, 'step': 5052}, {'eval_loss': 0.2778060734272003, 'eval_accuracy': 0.9054194506310319, 'eval_f1': 0.9123675883890494, 'eval_precision': 0.9089912280701754, 'eval_recall': 0.9157691245512289, 'eval_runtime': 11.6003, 'eval_samples_per_second': 580.589, 'eval_steps_per_second': 18.189, 'epoch': 3.0, 'step': 5052}, {'loss': 0.2943, 'grad_norm': 5.23865270614624, 'learning_rate': 1.2001187648456058e-05, 'epoch': 4.0, 'step': 6736}, {'eval_loss': 0.2678532004356384, 'eval_accuracy': 0.910022271714922, 'eval_f1': 0.9148637257656645, 'eval_precision': 0.9310837861023734, 'eval_recall': 0.8991991162662248, 'eval_runtime': 11.9067, 'eval_samples_per_second': 565.646, 'eval_steps_per_second': 17.721, 'epoch': 4.0, 'step': 6736}, {'loss': 0.2848, 'grad_norm': 3.4130496978759766, 'learning_rate': 1.0001187648456059e-05, 'epoch': 5.0, 'step': 8420}, {'eval_loss': 0.267210453748703, 'eval_accuracy': 0.910022271714922, 'eval_f1': 0.9168724279835391, 'eval_precision': 0.910874897792314, 'eval_recall': 0.9229494614747308, 'eval_runtime': 12.0494, 'eval_samples_per_second': 558.951, 'eval_steps_per_second': 17.511, 'epoch': 5.0, 'step': 8420}, {'loss': 0.2775, 'grad_norm': 2.2386434078216553, 'learning_rate': 8.001187648456058e-06, 'epoch': 6.0, 'step': 10104}, {'eval_loss': 0.2572481632232666, 'eval_accuracy': 0.9132887899034893, 'eval_f1': 0.9187308655719455, 'eval_precision': 0.9259467040673212, 'eval_recall': 0.9116266224799779, 'eval_runtime': 12.0053, 'eval_samples_per_second': 561.001, 'eval_steps_per_second': 17.576, 'epoch': 6.0, 'step': 10104}, {'loss': 0.2723, 'grad_norm': 2.5065760612487793, 'learning_rate': 6.001187648456057e-06, 'epoch': 7.0, 'step': 11788}, {'eval_loss': 0.2546166479587555, 'eval_accuracy': 0.9155159613956941, 'eval_f1': 0.9203192830135836, 'eval_precision': 0.9335227272727272, 'eval_recall': 0.9074841204087268, 'eval_runtime': 11.6744, 'eval_samples_per_second': 576.901, 'eval_steps_per_second': 18.074, 'epoch': 7.0, 'step': 11788}, {'loss': 0.2687, 'grad_norm': 3.593709945678711, 'learning_rate': 4.001187648456058e-06, 'epoch': 8.0, 'step': 13472}, {'eval_loss': 0.25229528546333313, 'eval_accuracy': 0.9150705270972531, 'eval_f1': 0.9208852005532503, 'eval_precision': 0.9224161817678027, 'eval_recall': 0.9193592930129798, 'eval_runtime': 11.7728, 'eval_samples_per_second': 572.083, 'eval_steps_per_second': 17.923, 'epoch': 8.0, 'step': 13472}, {'loss': 0.2655, 'grad_norm': 2.035546064376831, 'learning_rate': 2.001187648456057e-06, 'epoch': 9.0, 'step': 15156}, {'eval_loss': 0.25151702761650085, 'eval_accuracy': 0.9170007423904974, 'eval_f1': 0.9223934471747883, 'eval_precision': 0.9274148520379676, 'eval_recall': 0.9174261253797293, 'eval_runtime': 11.6479, 'eval_samples_per_second': 578.218, 'eval_steps_per_second': 18.115, 'epoch': 9.0, 'step': 15156}, {'loss': 0.2635, 'grad_norm': 2.518160581588745, 'learning_rate': 1.1876484560570071e-09, 'epoch': 10.0, 'step': 16840}, {'eval_loss': 0.25082287192344666, 'eval_accuracy': 0.9168522642910171, 'eval_f1': 0.9220923761825265, 'eval_precision': 0.929072049341183, 'eval_recall': 0.9152167909417288, 'eval_runtime': 11.5215, 'eval_samples_per_second': 584.56, 'eval_steps_per_second': 18.314, 'epoch': 10.0, 'step': 16840}, {'train_runtime': 2036.7204, 'train_samples_per_second': 264.538, 'train_steps_per_second': 8.268, 'total_flos': 5739742381421400.0, 'train_loss': 0.2955606945336856, 'epoch': 10.0, 'step': 16840}, {'eval_loss': 0.25151702761650085, 'eval_accuracy': 0.9170007423904974, 'eval_f1': 0.9223934471747883, 'eval_precision': 0.9274148520379676, 'eval_recall': 0.9174261253797293, 'eval_runtime': 11.2976, 'eval_samples_per_second': 596.147, 'eval_steps_per_second': 18.677, 'epoch': 10.0, 'step': 16840}, {'eval_loss': 0.2650555968284607, 'eval_accuracy': 0.9097253155159614, 'eval_f1': 0.9195128408790045, 'eval_precision': 0.9224435590969455, 'eval_recall': 0.9166006861968857, 'eval_runtime': 12.1809, 'eval_samples_per_second': 552.916, 'eval_steps_per_second': 17.322, 'epoch': 10.0, 'step': 16840}]
```

=== Summary over ranks (Sparse LoRA) ===

Rank 2:

Params: 665,858 / 67,620,868 (0.98%)
Train time: 2040.75 s
Val F1: 0.9072, Acc: 0.9004
Test F1: 0.9028, Acc: 0.8915
LoRA sparsity (<1e-3): 15.55%

Rank 4:

Params: 739,586 / 67,694,596 (1.09%)
Train time: 2031.34 s
Val F1: 0.9135, Acc: 0.9068
Test F1: 0.9064, Acc: 0.8950
LoRA sparsity (<1e-3): 21.52%

Rank 8:

Params: 887,042 / 67,842,052 (1.31%)
Train time: 2055.44 s
Val F1: 0.9178, Acc: 0.9121
Test F1: 0.9122, Acc: 0.9017
LoRA sparsity ($<1e-3$): 27.88%

Rank 16:

Params: 1,181,954 / 68,136,964 (1.73%)
Train time: 2037.07 s
Val F1: 0.9224, Acc: 0.9170
Test F1: 0.9195, Acc: 0.9097
LoRA sparsity ($<1e-3$): 35.68%

```
In [39]: # ===== SPARSE LoRA MODEL =====

from typing import Dict, Any, List, Optional
import math
from peft import LoraConfig, get_peft_model

# ----- Sparse LoRA config -----
RANKS: List[int] = [2, 4, 8, 16]
L1_LAMBDA = 1e-5 # sparsity strength for LoRA weights

def count_trainable_params(model: torch.nn.Module) -> int:
    return sum(p.numel() for p in model.parameters() if p.requires_grad)

def count_total_params(model: torch.nn.Module) -> int:
    return sum(p.numel() for p in model.parameters())

def compute_lora_sparsity(model: torch.nn.Module, threshold: float = 1e-3) -> float:
    """
    Approximate sparsity: fraction of LoRA parameters with |w| < threshold.
    """
    total = 0
    near_zero = 0
    for name, param in model.named_parameters():
        if "lora_" in name and param.requires_grad:
            data = param.detach().abs()
            total += data.numel()
            near_zero += (data < threshold).sum().item()
    return near_zero / total if total > 0 else math.nan

class SparseLoraTrainer(Trainer):
    """
    Trainer with L1 penalty only on LoRA parameters.
    """
    def __init__(self, l1_lambda: float = 0.0, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.l1_lambda = l1_lambda

    def compute_loss(
        self,
        model,
        inputs,
        return_outputs: bool = False,
        num_items_in_batch: Optional[int] = None,
    ):
        outputs = model(*inputs)
        loss = outputs.loss

        if self.l1_lambda > 0:
            l1_reg = 0.0
            for name, param in model.named_parameters():
                if "lora_" in name and param.requires_grad:
                    l1_reg = l1_reg + param.abs().sum()
            loss = loss + self.l1_lambda * l1_reg
```

```

        return (loss, outputs) if return_outputs else loss

results_per_rank: List[Dict[str, Any]] = []

for r in RANKS:
    print("\n" + "=" * 80)
    print(f"Training Sparse LoRA DistilBERT with rank = {r}, epochs = {NUM_EPOCHS}")
    print("=" * 80)

    set_all_seeds(SEED)

    # Base DistilBERT for this rank
    base_model = DistilBertForSequenceClassification.from_pretrained(
        "distilbert-base-uncased",
        num_labels=2,
    )

    # LoRA config: attention projections in DistilBERT
    lora_config = LoraConfig(
        r=r,
        lora_alpha=2 * r,
        lora_dropout=0.1,
        bias="none",
        task_type="SEQ_CLS",  # sequence classification
        target_modules=["q_lin", "k_lin", "v_lin", "out_lin"],
    )

    lora_model = get_peft_model(base_model, lora_config)
    lora_model.to(DEVICE)

    total_params = count_total_params(lora_model)
    trainable_params = count_trainable_params(lora_model)
    param_ratio = trainable_params / total_params

    print(f"[Rank {r}] total params: {total_params:,}")
    print(f"[Rank {r}] trainable params: {trainable_params:,}")
    print(f"[Rank {r}] trainable params ratio (trainable / total): {param_ratio:.4%}")

    output_dir = f"./sparse_lora_rank{r}"
    os.makedirs(output_dir, exist_ok=True)

    training_args_lora = TrainingArguments(
        output_dir=output_dir,
        num_train_epochs=NUM_EPOCHS,
        per_device_train_batch_size=BATCH_SIZE,
        per_device_eval_batch_size=BATCH_SIZE,
        learning_rate=LR,
        weight_decay=0.01,
        eval_strategy="epoch",
        save_strategy="epoch",
        logging_strategy="epoch",
        load_best_model_at_end=True,
        metric_for_best_model="f1",
        greater_is_better=True,
        seed=SEED,
        report_to="none",
    )

    trainer = SparseLoraTrainer(
        ll_lambda=L1_LAMBDA,
        model=lora_model,
        args=training_args_lora,
        train_dataset=train_ds,
        eval_dataset=val_ds,
        tokenizer=tokenizer,
        data_collator=collator,
        compute_metrics=compute_metrics,
    )

    start_time = time.time()
    trainer.train()

```

```

end_time = time.time()
train_time = end_time - start_time
print(f"[Rank {r}] Training time: {train_time:.2f} seconds")

# --- final evals ---
val_metrics = trainer.evaluate(eval_dataset=val_ds)
test_metrics = trainer.evaluate(eval_dataset=test_ds)
lora_sparsity = compute_lora_sparsity(lora_model, threshold=1e-3)

print(f"[Rank {r}] Validation metrics: {val_metrics}")
print(f"[Rank {r}] Test metrics: {test_metrics}")
print(f"[Rank {r}] LoRA sparsity (<1e-3): {lora_sparsity:.2%}")

# =====
# SAVE METRICS / MODEL / LOG
# =====
# 1) save metrics
metrics_payload = {
    "rank": r,
    "train_time_sec": train_time,
    "total_params": int(total_params),
    "trainable_params": int(trainable_params),
    "param_ratio": float(param_ratio),
    "lora_sparsity_<1e-3": float(lora_sparsity),
    "val_metrics": val_metrics,
    "test_metrics": test_metrics,
}
with open(os.path.join(output_dir, "final_metrics.json"), "w") as f:
    json.dump(metrics_payload, f, indent=4)

# 2) save final model (best checkpoint)
final_model_dir = os.path.join(output_dir, "final_model")
trainer.save_model(final_model_dir) # saves model + config
tokenizer.save_pretrained(final_model_dir) # save tokenizer too
print(f"[Rank {r}] Saved model to {final_model_dir}")

# 3) save convergence history
log_history = trainer.state.log_history
with open(os.path.join(output_dir, "log_history.json"), "w") as f:
    json.dump(log_history, f, indent=4)
print(f"[Rank {r}] Saved log history to log_history.json")

# --- store in-memory summary for printing ---
results_per_rank.append(
    {
        "rank": r,
        "total_params": total_params,
        "trainable_params": trainable_params,
        "param_ratio": param_ratio,
        "train_time_sec": train_time,
        "val_metrics": val_metrics,
        "test_metrics": test_metrics,
        "lora_sparsity(<1e-3)": lora_sparsity,
    }
)

print("\n\n=== Summary over ranks (Sparse LoRA) ===")
for res in results_per_rank:
    r = res["rank"]
    print(f"\nRank {r}:")
    print(f"  Params: {res['trainable_params']:,} / {res['total_params']:,} "
          f"f{res['param_ratio']:.2%}")
    print(f"  Train time: {res['train_time_sec']:.2f} s")
    print(f"  Val F1: {res['val_metrics'].get('eval_f1', float('nan')):.4f}, "
          f"fAcc: {res['val_metrics'].get('eval_accuracy', float('nan')):.4f}")
    print(f"  Test F1: {res['test_metrics'].get('eval_f1', float('nan')):.4f}, "
          f"fAcc: {res['test_metrics'].get('eval_accuracy', float('nan')):.4f}")
    print(f"  LoRA sparsity (<1e-3): {res['lora_sparsity(<1e-3)']:.2%}")

```

```

=====
Training Sparse LoRA DistilBERT with rank = 2, epochs = 10
=====

```

Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight', 'pre_classifier.bias', 'pre_classifier.weight']

You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

[Rank 2] total params: 67,620,868

[Rank 2] trainable params: 665,858

[Rank 2] trainable params ratio (trainable / total): 0.9847%

[16840/16840 33:19, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.393900	0.296121	0.876466	0.882320	0.904320	0.861364
2	0.316000	0.280038	0.885375	0.892867	0.897350	0.888429
3	0.302800	0.270784	0.891314	0.898951	0.898703	0.899199
4	0.292400	0.262972	0.894432	0.900989	0.908708	0.893400
5	0.285900	0.261988	0.896659	0.904422	0.899481	0.909417
6	0.281500	0.254590	0.897847	0.904205	0.911823	0.896714
7	0.277900	0.252679	0.899183	0.904995	0.917187	0.893123
8	0.275900	0.251038	0.899926	0.906957	0.906707	0.907208
9	0.272700	0.250182	0.900371	0.907231	0.908361	0.906103
10	0.271600	0.249398	0.900371	0.906793	0.912241	0.901408

[Rank 2] Training time: 2000.22 seconds

[211/211 00:21]

[Rank 2] Validation metrics: {'eval_loss': 0.25018182396888733, 'eval_accuracy': 0.9003711952487008, 'eval_f1': 0.9072307479607356, 'eval_precision': 0.9083610188261351, 'eval_recall': 0.9061032863849765, 'eval_runtime': 11.2932, 'eval_samples_per_second': 596.375, 'eval_steps_per_second': 18.684, 'epoch': 10.0}

[Rank 2] Test metrics: {'eval_loss': 0.26539742946624756, 'eval_accuracy': 0.8914625092798812, 'eval_f1': 0.9028313172936329, 'eval_precision': 0.9094804499196572, 'eval_recall': 0.8962787015043547, 'eval_runtime': 10.8577, 'eval_samples_per_second': 620.299, 'eval_steps_per_second': 19.433, 'epoch': 10.0}

[Rank 2] LoRA sparsity (<1e-3): 15.55%

[Rank 2] Saved model to ./sparse_lora_rank2/final_model

[Rank 2] Saved log history to log_history.json

=====
Training Sparse LoRA DistilBERT with rank = 4, epochs = 10
=====

Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight', 'pre_classifier.bias', 'pre_classifier.weight']

You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

[Rank 4] total params: 67,694,596

[Rank 4] trainable params: 739,586

[Rank 4] trainable params ratio (trainable / total): 1.0925%

[16840/16840 34:11, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.386300	0.294177	0.879733	0.885915	0.903995	0.868545
2	0.312500	0.276367	0.890275	0.897801	0.899169	0.896437
3	0.298700	0.267596	0.894878	0.902399	0.900908	0.903894
4	0.287600	0.258988	0.899926	0.905761	0.917304	0.894504
5	0.280500	0.258202	0.902004	0.909564	0.902638	0.916598
6	0.275400	0.249693	0.904083	0.910103	0.917251	0.903065
7	0.271600	0.247551	0.905865	0.911477	0.921774	0.901408
8	0.268800	0.245857	0.906756	0.913451	0.911692	0.915217
9	0.265800	0.244966	0.906607	0.913062	0.913946	0.912179
10	0.265100	0.244083	0.907053	0.913152	0.917480	0.908865

[Rank 4] Training time: 2051.59 seconds

[211/211 00:22]

[Rank 4] Validation metrics: {'eval_loss': 0.245857372879982, 'eval_accuracy': 0.9067557535263548, 'eval_f1': 0.9134509371554576, 'eval_precision': 0.9116918844566713, 'eval_recall': 0.9152167909417288, 'eval_runtime': 11.4603, 'eval_samples_per_second': 587.683, 'eval_steps_per_second': 18.411, 'epoch': 10.0}

[Rank 4] Test metrics: {'eval_loss': 0.2627865672111511, 'eval_accuracy': 0.895025983667409, 'eval_f1': 0.9063948100092678, 'eval_precision': 0.9094048884165781, 'eval_recall': 0.9034045922406968, 'eval_runtime': 11.9928, 'eval_samples_per_second': 561.588, 'eval_steps_per_second': 17.594, 'epoch': 10.0}

[Rank 4] LoRA sparsity (<1e-3): 21.52%

[Rank 4] Saved model to ./sparse_lora_rank4/final_model

[Rank 4] Saved log history to log_history.json

=====
Training Sparse LoRA DistilBERT with rank = 8, epochs = 10
=====

Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight', 'pre_classifier.bias', 'pre_classifier.weight']

You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

[Rank 8] total params: 67,842,052

[Rank 8] trainable params: 887,042

[Rank 8] trainable params ratio (trainable / total): 1.3075%

[16840/16840 33:55, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy	F1	Precision	Recall
1	0.386200	0.295880	0.884781	0.891134	0.905617	0.877106
2	0.313800	0.277604	0.894878	0.902237	0.902237	0.902237
3	0.299300	0.268710	0.897847	0.905546	0.900355	0.910798
4	0.287800	0.259213	0.904826	0.910362	0.922096	0.898923
5	0.280200	0.259396	0.904380	0.912094	0.901754	0.922673
6	0.273700	0.249641	0.908389	0.914222	0.920493	0.908036
7	0.269600	0.247296	0.911359	0.916468	0.928815	0.904446
8	0.266700	0.245347	0.910171	0.916632	0.914741	0.918531
9	0.263500	0.244415	0.911210	0.917220	0.919512	0.914941
10	0.262700	0.243555	0.912101	0.917823	0.922690	0.913007

[Rank 8] Training time: 2036.44 seconds

[211/211 00:22]


```
[Rank 16] Validation metrics: {'eval_loss': 0.25151702761650085, 'eval_accuracy': 0.9170007423904974, 'eval_f1': 0.9223934471747883, 'eval_precision': 0.9274148520379676, 'eval_recall': 0.9174261253797293, 'eval_runtime': 11.2259, 'eval_samples_per_second': 599.952, 'eval_steps_per_second': 18.796, 'epoch': 10.0}
[Rank 16] Test metrics: {'eval_loss': 0.2650555968284607, 'eval_accuracy': 0.9097253155159614, 'eval_f1': 0.9195128408790045, 'eval_precision': 0.9224435590969455, 'eval_recall': 0.9166006861968857, 'eval_runtime': 10.8663, 'eval_samples_per_second': 619.804, 'eval_steps_per_second': 19.418, 'epoch': 10.0}
[Rank 16] LoRA sparsity (<1e-3): 35.68%
[Rank 16] Saved model to ./sparse_lora_rank16/final_model
[Rank 16] Saved log history to log_history.json
```

=== Summary over ranks (Sparse LoRA) ===

Rank 2:

```
Params: 665,858 / 67,620,868 (0.98%)
Train time: 2000.22 s
Val F1: 0.9072, Acc: 0.9004
Test F1: 0.9028, Acc: 0.8915
LoRA sparsity (<1e-3): 15.55%
```

Rank 4:

```
Params: 739,586 / 67,694,596 (1.09%)
Train time: 2051.59 s
Val F1: 0.9135, Acc: 0.9068
Test F1: 0.9064, Acc: 0.8950
LoRA sparsity (<1e-3): 21.52%
```

Rank 8:

```
Params: 887,042 / 67,842,052 (1.31%)
Train time: 2036.44 s
Val F1: 0.9178, Acc: 0.9121
Test F1: 0.9122, Acc: 0.9017
LoRA sparsity (<1e-3): 27.88%
```

Rank 16:

```
Params: 1,181,954 / 68,136,964 (1.73%)
Train time: 2026.30 s
Val F1: 0.9224, Acc: 0.9170
Test F1: 0.9195, Acc: 0.9097
LoRA sparsity (<1e-3): 35.68%
```

IMDB Dataset

Have commented out in the corresponding section

Just to replace the loaded dataset accordingly:

- DATASET = "imdb"
- TEXT_COL = "text"
- LABEL_COL = "label"

In []: