```
!pip install -q "transformers>=4.40" "datasets>=2.19" accelerate peft bitsandbytes scikit-learn matplotlib pandas
!pip install -U transformers

import os
import time
import random
import math
import copy
import numpy as np
import torch

from datasets import load_dataset
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
import matplotlib.pyplot as plt
import pandas as pd

from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    Trainer,
    TrainingArguments,
    DataCollatorWithPadding,
    BitsAndBytesConfig,
)

from peft import (
    LoraConfig,
    TaskType,
    get_peft_model,
    prepare_model_for_kbit_training,
)

from torch.quantization import quantize_dynamic
from torch.nn.utils import prune

# GLOBALS: SEED, DEVICE

SEED = 42

def set_seed(seed=SEED):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)

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

```
Requirement already satisfied: transformers in /usr/local/lib/python3.12/dist-packages (4.57.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from transformers) (3.20.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.34.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.36
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.12/dist-packages (from transformers) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (25.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from transformers) (6.0.3)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/dist-packages (from transformers) (2025.11.3)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from transformers) (2.32.4)
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.22.
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.7.0)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.12/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub<1.0,>=0.34.0->t
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub<1.0,>
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub<1.0,>=0.34.
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests->transformers)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->transformers) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->transformers) (2.5.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->transformers) (2025
'cuda'
```

```
# 1. CONFIG & FLAGS


# Choose dataset here: "imdb" or "ag_news"
```

```
DATASET_NAME = "imdb"   # change to "ag_news" for news classification

# Small, fast debug run vs full run
DEBUG = False

CONFIG = {
    "model_name": "distilbert-base-uncased",
    "max_length": 256,
    "train_samples": 4000,
    "valid_samples": 1000,
    "num_epochs": 2,
    "train_batch_size": 16,
    "eval_batch_size": 32,
    "learning_rate": 2e-5,
    "weight_decay": 0.01,
    "warmup_ratio": 0.1,
    "lora_r": 8,
    "lora_alpha": 16,
    "lora_dropout": 0.05,
}

if DEBUG:
    CONFIG["train_samples"] = 512
    CONFIG["valid_samples"] = 256
    CONFIG["num_epochs"]   = 1
    CONFIG["train_batch_size"] = 8
    CONFIG["eval_batch_size"]  = 16

# Tokenizer is shared across datasets
tokenizer = AutoTokenizer.from_pretrained(CONFIG["model_name"], use_fast=True)
```

```
# HELPER FUNCTIONS: GPU STATS, LATENCY, METRICS


def reset_gpu_memory_stats():
    if torch.cuda.is_available():
        torch.cuda.reset_peak_memory_stats()
        torch.cuda.empty_cache()

def get_max_gpu_memory_gb():
    if not torch.cuda.is_available():
        return None
    return torch.cuda.max_memory_allocated() / (1024 ** 3)

def measure_inference_latency(trainer, eval_dataset, num_batches=20):
    """
    Approximate latency using the trainer's prediction loop.
    Measures average time per batch and per example.
    """
    dataloader = trainer.get_eval_dataloader(eval_dataset)
    dataloader_iter = iter(dataloader)

    total_examples = 0
    total_time = 0.0

    trainer.model.eval()
    for i in range(num_batches):
        try:
            batch = next(dataloader_iter)
        except StopIteration:
            break
        for k in batch:
            batch[k] = batch[k].to(trainer.model.device)

        with torch.no_grad():
            start = time.perf_counter()
            _ = trainer.model(**batch)
            if torch.cuda.is_available():
                torch.cuda.synchronize()
            end = time.perf_counter()

        bsz = batch["input_ids"].shape[0]
        total_examples += bsz
        total_time += (end - start)
```

```
    if total_examples == 0 or num_batches == 0:
        return None, None

    avg_batch_latency = total_time / num_batches
    avg_example_latency = total_time / total_examples
    return avg_batch_latency, avg_example_latency

def compute_metrics(p):
    preds = np.argmax(p.predictions, axis=-1)
    labels = p.label_ids
    acc = accuracy_score(labels, preds)
    f1 = f1_score(labels, preds, average="weighted")
    return {"accuracy": acc, "f1": f1}
```

```
# DATA LOADING: IMDB + AG NEWS


def load_tokenized_dataset(dataset_name):
    """
    Returns:
      train_subset, test_subset, num_labels
    """
    raw_ds = load_dataset(dataset_name)

    # Both IMDB and AG News use 'text' + 'label'
    text_column = "text"

    def tokenize_fn(examples):
        return tokenizer(
            examples[text_column],
            truncation=True,
            max_length=CONFIG["max_length"],
            padding=False,
        )

    tokenized_ds = raw_ds.map(tokenize_fn, batched=True, remove_columns=[text_column])
    tokenized_ds = tokenized_ds.rename_column("label", "labels")

    num_labels = raw_ds["train"].features["labels"].num_classes if "labels" in raw_ds["train"].features else raw_ds["train"].fe

    train_subset = tokenized_ds["train"].shuffle(seed=SEED).select(
        range(CONFIG["train_samples"])
    )
    test_subset = tokenized_ds["test"].shuffle(seed=SEED).select(
        range(CONFIG["valid_samples"])
    )

    return train_subset, test_subset, num_labels

train_subset, test_subset, num_labels = load_tokenized_dataset(DATASET_NAME)
CONFIG["num_labels"] = num_labels
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

len(train_subset), len(test_subset), num_labels
```

```
(4000, 1000, 2)
```

```
# BASELINE MODEL — FULL FINE-TUNING


set_seed()
reset_gpu_memory_stats()

baseline_model = AutoModelForSequenceClassification.from_pretrained(
    CONFIG["model_name"],
    num_labels=CONFIG["num_labels"],
).to(device)

baseline_args = TrainingArguments(
    output_dir=f"{DATASET_NAME}_baseline_fp16",
    learning_rate=CONFIG["learning_rate"],
```

```
    per_device_train_batch_size=CONFIG["train_batch_size"],
    per_device_eval_batch_size=CONFIG["eval_batch_size"],
    num_train_epochs=CONFIG["num_epochs"],
    weight_decay=CONFIG["weight_decay"],
    logging_steps=50,
    fp16=torch.cuda.is_available(),
    report_to="none",
)

baseline_trainer = Trainer(
    model=baseline_model,
    args=baseline_args,
    train_dataset=train_subset,
    eval_dataset=test_subset,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)

print(f"Training Baseline FP16 on {DATASET_NAME}...")
train_result = baseline_trainer.train()
baseline_metrics_train = train_result.metrics
baseline_metrics_eval = baseline_trainer.evaluate()

baseline_batch_latency, baseline_example_latency = measure_inference_latency(
    baseline_trainer, test_subset
)
baseline_max_vram_gb = get_max_gpu_memory_gb()

baseline_summary = {
    "dataset": DATASET_NAME,
    "config": "baseline_fp16",
    "accuracy": baseline_metrics_eval["eval_accuracy"],
    "f1": baseline_metrics_eval["eval_f1"],
    "batch_latency_s": baseline_batch_latency,
    "example_latency_s": baseline_example_latency,
    "max_vram_gb": baseline_max_vram_gb,
}

print("Baseline summary:", baseline_summary)
```

```
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased an
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
/tmp/ipython-input-3420760721.py:24: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.
  baseline_trainer = Trainer(
Training Baseline FP16 on imdb...
```

[500/500 01:21, Epoch 2/2]

Step	Training Loss
50	0.643700
100	0.356400
150	0.374800
200	0.329200
250	0.336100
300	0.278600
350	0.241300
400	0.192400
450	0.205900
500	0.167800

[32/32 00:02]

```
Baseline summary: {'dataset': 'imdb', 'config': 'baseline_fp16', 'accuracy': 0.88, 'f1': 0.880016801680168, 'batch_latency_s': 0
```

```
#LoRA HELPERS + TRAINING (r=8, r=4)


def make_lora_config(r):
    return LoraConfig(
```

```
        task_type=TaskType.SEQ_CLS,
        r=r,
        lora_alpha=CONFIG["lora_alpha"],
        lora_dropout=CONFIG["lora_dropout"],
        target_modules=["q_lin", "k_lin", "v_lin", "out_lin"],  # DistilBERT-safe
    )


def load_lora_model(r, output_dir):
    """
    DistilBERT + LoRA in standard precision.
    This avoids the HF/bitsandbytes quantization bug.
    """
    reset_gpu_memory_stats()
    set_seed()

    print(f"Using standard precision + LoRA (r={r})...")
    model = AutoModelForSequenceClassification.from_pretrained(
        CONFIG["model_name"],
        num_labels=CONFIG["num_labels"],
    ).to(device)

    # Attach LoRA adapters
    lora_cfg = make_lora_config(r)
    model = get_peft_model(model, lora_cfg)

    # Training hyperparams shared with baseline
    args = TrainingArguments(
        output_dir=output_dir,
        learning_rate=CONFIG["learning_rate"],
        per_device_train_batch_size=CONFIG["train_batch_size"],
        per_device_eval_batch_size=CONFIG["eval_batch_size"],
        num_train_epochs=CONFIG["num_epochs"],
        weight_decay=CONFIG["weight_decay"],
        logging_steps=10,
        fp16=torch.cuda.is_available(),
        report_to="none",
    )

    trainer = Trainer(
        model=model,
        args=args,
        train_dataset=train_subset,
        eval_dataset=test_subset,
        tokenizer=tokenizer,
        data_collator=data_collator,
        compute_metrics=compute_metrics,
    )

    return model, trainer
```

```
# LoRA r = 8
lora_model, lora_trainer = load_lora_model(
    r=8, output_dir=f"{DATASET_NAME}_lora_r8"
)

print("Training LoRA (r=8)...")
lora_train_result = lora_trainer.train()
lora_metrics_eval = lora_trainer.evaluate()

lora_batch_latency, lora_example_latency = measure_inference_latency(
    lora_trainer, test_subset
)
lora_max_vram_gb = get_max_gpu_memory_gb()

lora_summary = {
    "dataset": DATASET_NAME,
    "config": "lora_r8",
    "accuracy": lora_metrics_eval["eval_accuracy"],
    "f1": lora_metrics_eval["eval_f1"],
    "batch_latency_s": lora_batch_latency,
    "example_latency_s": lora_example_latency,
    "max_vram_gb": lora_max_vram_gb,
```

```
}
print("LoRA r=8 summary:", lora_summary)
```

```
Using standard precision + LoRA (r=8)...
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased an
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
/tmp/ipython-input-3911773296.py:45: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.
  trainer = Trainer(
Training LoRA (r=8)...
```

[500/500 00:50, Epoch 2/2]

Step	Training Loss
10	0.699000
20	0.694100
30	0.697400
40	0.691300
50	0.684600
60	0.685100

```
# LoRA r = 4
del lora_trainer, lora_model
torch.cuda.empty_cache()
reset_gpu_memory_stats()

lora_r4_model, lora_r4_trainer = load_lora_model(
    r=4, output_dir=f"{DATASET_NAME}_lora_r4"
)

print("Training LoRA (r=4)...")
lora_r4_train_result = lora_r4_trainer.train()
lora_r4_metrics_eval = lora_r4_trainer.evaluate()

lora_r4_batch_latency, lora_r4_example_latency = measure_inference_latency(
    lora_r4_trainer, test_subset
)
lora_r4_max_vram_gb = get_max_gpu_memory_gb()

lora_r4_summary = {
    "dataset": DATASET_NAME,
    "config": "lora_r4",
    "accuracy": lora_r4_metrics_eval["eval_accuracy"],
    "f1": lora_r4_metrics_eval["eval_f1"],
    "batch_latency_s": lora_r4_batch_latency,
    "example_latency_s": lora_r4_example_latency,
    "max_vram_gb": lora_r4_max_vram_gb,
}
print("LoRA r=4 summary:", lora_r4_summary)
```

260	0.596600
270	0.575800
280	0.570300
290	0.579800
300	0.554300
310	0.519200
320	0.532800
330	0.535200
340	0.519500
350	0.484100
360	0.469000
370	0.476700
380	0.454700
390	0.432600
400	0.451000
410	0.423400
420	0.431800

430	0.442500
440	0.410700
450	0.423100
460	0.430600
470	0.362300
480	0.389400
490	0.430900
500	0.414400

[32/32 00:02]

```
LoRA r=8 summary: {'dataset': 'imdb', 'config': 'lora_r8', 'accuracy': 0.835, 'f1': 0.8349363970864309, 'batch_latency_s': 0.057
```

```
Using standard precision + LoRA (r=4)...
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased an
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
/tmp/ipython-input-3911773296.py:45: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.
  trainer = Trainer(
Training LoRA (r=4)...
```

[500/500 00:48, Epoch 2/2]

Step	Training Loss
10	0.699000
20	0.694100
30	0.697400
40	0.691300
50	0.684600

```
# 6. Dynamic INT8 Quantization Baseline (CPU-friendly)


print("\nEvaluating dynamic int8 quantized baseline (CPU-only)...")
reset_gpu_memory_stats()
set_seed()

baseline_cpu_for_quant = copy.deepcopy(baseline_model).to("cpu").eval()

quant_model = quantize_dynamic(
    baseline_cpu_for_quant,
    {torch.nn.Linear},
    dtype=torch.qint8,
)

quant_args = TrainingArguments(
    output_dir=f"{DATASET_NAME}_baseline_dynamic_int8",
    per_device_eval_batch_size=CONFIG["eval_batch_size"],
    fp16=False,        # no mixed precision
    no_cuda=True,      # <-- IMPORTANT: force CPU
    report_to="none",
)

quant_trainer = Trainer(
    model=quant_model,          # stays on CPU
    args=quant_args,
    eval_dataset=test_subset,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)


quant_metrics_eval = quant_trainer.evaluate()
quant_batch_latency, quant_example_latency = measure_inference_latency(
    quant_trainer, test_subset
)
quant_max_vram_gb = get_max_gpu_memory_gb()

quant_summary = {
    "dataset": DATASET_NAME,
    "config": "baseline_dynamic_int8",
    "accuracy": quant_metrics_eval["eval_accuracy"],
    "f1": quant_metrics_eval["eval_f1"],
    "batch_latency_s": quant_batch_latency,
    "example_latency_s": quant_example_latency,
    "max_vram_gb": quant_max_vram_gb,
}
print("Dynamic int8 summary:", quant_summary)
```

380	0.441800
390	0.421600
400	0.442900
410	0.412200
420	0.424400

```
430   0.435300
440   0.405400
450   0.414100
460   0.424600
470   0.355700
480   0.381600
490   0.428000
```

```
Evaluating dynamic int8 quantized baseline (CPU-only)...
/tmp/ipython-input-3843088670.py:10: DeprecationWarning: torch.ao.quantization is deprecated and will be removed in 2.10.
For migrations of users:
1. Eager mode quantization (torch.ao.quantization.quantize, torch.ao.quantization.quantize_dynamic), please migrate to use torch
2. FX graph mode quantization (torch.ao.quantization.quantize_fx.prepare_fx,torch.ao.quantization.quantize_fx.convert_fx, please
3. pt2e quantization has been migrated to torchao (https://github.com/pytorch/ao/tree/main/torchao/quantization/pt2e)
see https://github.com/pytorch/ao/issues/2259 for more details
  quant_model = quantize_dynamic(
/usr/local/lib/python3.12/dist-packages/transformers/training_args.py:1636: FutureWarning: using `no_cuda` is deprecated and wil
  warnings.warn(
/tmp/ipython-input-3843088670.py:24: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.
  quant_trainer = Trainer(
[32/32 05:13]
Dynamic int8 summary: {'dataset': 'imdb', 'config': 'baseline_dynamic_int8', 'accuracy': 0.851, 'f1': 0.8498343156575335, 'batch
[32/32 00:02]
LoRA r=4 summary: {'dataset': 'imdb', 'config': 'lora_r4', 'accuracy': 0.832, 'f1': 0.8319104947544254, 'batch_latency_s': 0.057
```

```python
# 7. Pruning Baseline (magnitude pruning on Linear layers)


print("\nEvaluating pruned baseline (L1 unstructured pruning)...")
reset_gpu_memory_stats()
set_seed()

pruned_model = copy.deepcopy(baseline_model).to("cpu").eval()

# Amount of weights to prune
PRUNE_AMOUNT = 0.3

for name, module in pruned_model.named_modules():
    # prune all Linear layers;
    if isinstance(module, torch.nn.Linear):
        prune.l1_unstructured(module, name="weight", amount=PRUNE_AMOUNT)
        prune.remove(module, "weight")  # make pruning permanent

pruned_trainer = Trainer(
    model=pruned_model.to(device),
    args=TrainingArguments(
        output_dir=f"{DATASET_NAME}_baseline_pruned",
        per_device_eval_batch_size=CONFIG["eval_batch_size"],
        report_to="none",
    ),
    eval_dataset=test_subset,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)

pruned_metrics_eval = pruned_trainer.evaluate()
pruned_batch_latency, pruned_example_latency = measure_inference_latency(
    pruned_trainer, test_subset
)
pruned_max_vram_gb = get_max_gpu_memory_gb()

pruned_summary = {
    "dataset": DATASET_NAME,
    "config": f"baseline_pruned_{int(PRUNE_AMOUNT*100)}pct",
    "accuracy": pruned_metrics_eval["eval_accuracy"],
    "f1": pruned_metrics_eval["eval_f1"],
    "batch_latency_s": pruned_batch_latency,
    "example_latency_s": pruned_example_latency,
    "max_vram_gb": pruned_max_vram_gb,
}
print("Pruned baseline summary:", pruned_summary)
```

```
Evaluating pruned baseline (L1 unstructured pruning)...
/tmp/ipython-input-2775451064.py:19: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.
  pruned_trainer = Trainer(
[32/32 00:01]
Pruned baseline summary: {'dataset': 'imdb', 'config': 'baseline_pruned_30pct', 'accuracy': 0.874, 'f1': 0.8739455601606632, 'ba
```

Start coding or generate with AI.

```
#RESULTS TABLE
results = [
    baseline_summary,
    lora_summary,
    lora_r4_summary,
    quant_summary,
    pruned_summary,
]

df_results = pd.DataFrame(results)
df_results["throughput_samples_per_s"] = 1.0 / df_results["example_latency_s"]

print("\n=== Final Comparison Table ===")
display(df_results)
```

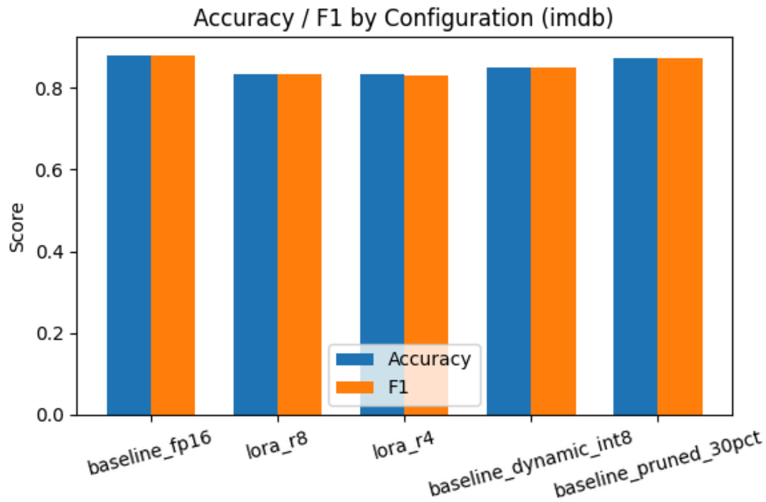
=== Final Comparison Table ===

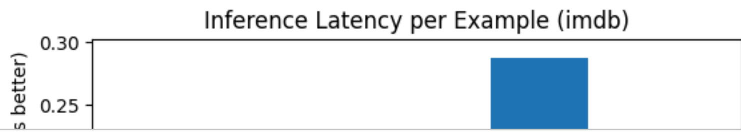
	dataset	config	accuracy	f1	batch_latency_s	example_latency_s	max_vram_gb	throughput_samples_per_s
0	imdb	baseline_fp16	0.880	0.880017	0.049228	0.001538	1.936972	650.039523
1	imdb	lora_r8	0.835	0.834936	0.057822	0.001807	2.094080	553.421907
2	imdb	lora_r4	0.832	0.831910	0.057221	0.001788	2.095575	559.236158
3	imdb	baseline_dynamic_int8	0.851	0.849834	9.212486	0.287890	1.540792	3.473547
4	imdb	baseline_pruned_30pct	0.874	0.873946	0.041558	0.001299	1.803930	770.016941

Next steps: Generate code with df_results New interactive sheet

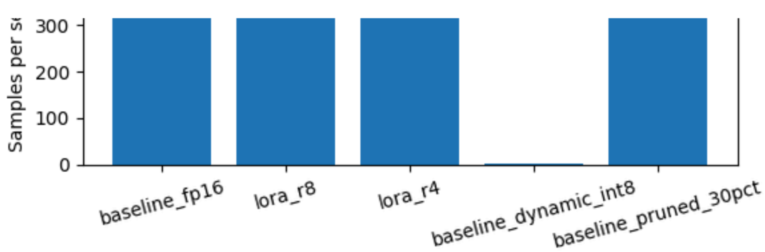
```
# VISUALIZATIONS


# Accuracy / F1
plt.figure(figsize=(6,4))
x = np.arange(len(df_results))
width = 0.35
plt.bar(x - width/2, df_results["accuracy"], width, label="Accuracy")
plt.bar(x + width/2, df_results["f1"], width, label="F1")
plt.xticks(x, df_results["config"], rotation=15)
plt.ylabel("Score")
plt.title(f"Accuracy / F1 by Configuration ({DATASET_NAME})")
plt.legend()
plt.tight_layout()
plt.show()

# Latency
plt.figure(figsize=(6,4))
plt.bar(df_results["config"], df_results["example_latency_s"])
plt.ylabel("Seconds per example (lower is better)")
plt.title(f"Inference Latency per Example ({DATASET_NAME})")
plt.xticks(rotation=15)
plt.tight_layout()
plt.show()

# Throughput
plt.figure(figsize=(6,4))
plt.bar(df_results["config"], df_results["throughput_samples_per_s"])
plt.ylabel("Samples per second (higher is better)")
plt.title(f"Throughput by Configuration ({DATASET_NAME})")
plt.xticks(rotation=15)
plt.tight_layout()
plt.show()

# Accuracy vs Latency trade-off
plt.figure(figsize=(6,4))
plt.scatter(df_results["example_latency_s"], df_results["accuracy"])

for i, row in df_results.iterrows():
    plt.annotate(
        row["config"],
        (row["example_latency_s"], row["accuracy"]),
        xytext=(5, 5),
```

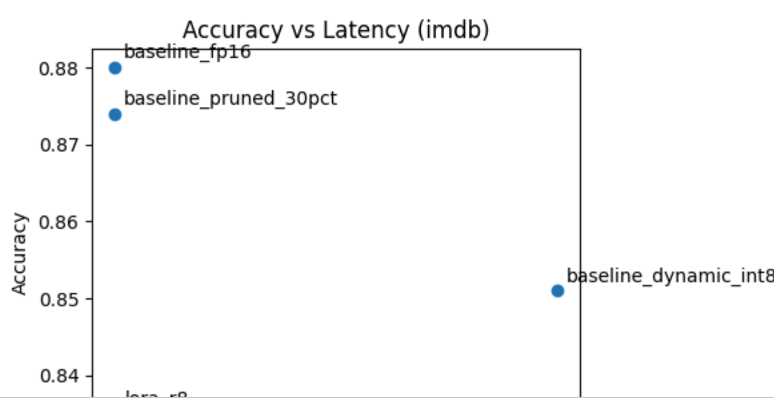
```
        textcoords="offset points",
    )

plt.xlabel("Seconds per example (lower is better)")
plt.ylabel("Accuracy")
plt.title(f"Accuracy vs Latency ({DATASET_NAME})")
plt.tight_layout()
plt.show()

# VRAM (if GPU is available)
vram_vals = pd.to_numeric(df_results["max_vram_gb"], errors="coerce")
non_na = vram_vals.dropna()

if non_na.empty or (non_na == 0).all():
    print("No GPU memory stats available (CPU-only run). VRAM plot skipped.")
else:
    plt.figure(figsize=(6,4))
    plt.bar(df_results.loc[non_na.index, "config"], non_na)
    plt.ylabel("Max VRAM (GB)")
    plt.title(f"Max GPU Memory Usage ({DATASET_NAME})")
    plt.xticks(rotation=15)
    plt.tight_layout()
    plt.show()
```

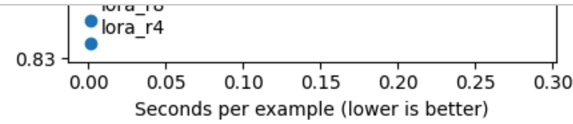


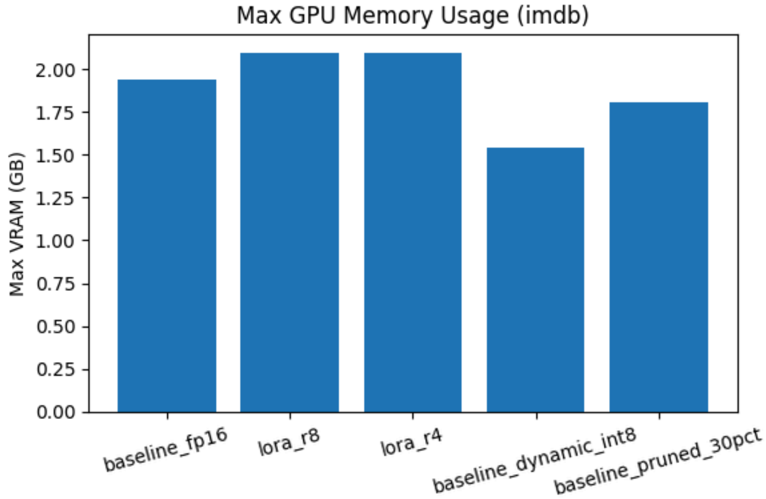



```
#Confusion Matrices to visualize results further

def plot_confusion_for_trainer(trainer, title, num_labels):
    preds_output = trainer.predict(test_subset)
    preds = np.argmax(preds_output.predictions, axis=-1)
    labels = preds_output.label_ids
    cm = confusion_matrix(labels, preds)

    plt.figure(figsize=(4,4))
    plt.imshow(cm, interpolation="nearest")
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(num_labels)
    plt.xticks(tick_marks, tick_marks)
    plt.yticks(tick_marks, tick_marks)
    plt.xlabel("Predicted label")
    plt.ylabel("True label")
    plt.tight_layout()
    plt.show()

plot_confusion_for_trainer(baseline_trainer, f"Baseline FP16 - {DATASET_NAME}", num_labels)
plot_confusion_for_trainer(lora_r4_trainer, f"LoRA r=4 - {DATASET_NAME}", num_labels)
plot_confusion_for_trainer(quant_trainer, f"Dynamic Int8 - {DATASET_NAME}", num_labels)
plot_confusion_for_trainer(pruned_trainer, f"Pruned Baseline - {DATASET_NAME}", num_labels)
```






lora_r4
0.83
0.00 0.05 0.10 0.15 0.20 0.25 0.30
Seconds per example (lower is better)


Max GPU Memory Usage (imdb)
Max VRAM (GB)
0.00 0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00
baseline_fp16
lora_r8
lora_r4
baseline_dynamic_int8
baseline_pruned_30pct