Assignment 4

```
# === IMPORTS AND SETUP ===
from datasets import load_dataset
from transformers import AutoTokenizer, AutoModelForSequenceClassification, TrainingArguments, Trainer, pipeline
import torch, numpy as np, evaluate, torch.nn as nn, torch.nn.functional as F, pandas as pd, matplotlib.pyplot as plt
from torch.quantization import quantize_dynamic
from time import perf_counter
from pathlib import Path
import optuna
```

```
# === DATASET AND MODEL CHECKPOINTS ===
clinc = load_dataset("clinc_oos", "plus")
teacher_ckpt = "answerdotai/ModernBERT-large"
student_ckpt = "bert-base-uncased"           #This is the chosen student model

# Get label mappings
intents = clinc["test"].features["intent"]
id2label = {i: label for i, label in enumerate(intents.names)}
label2id = {label: i for i, label in enumerate(intents.names)}
num_labels = intents.num_classes

# === TOKENIZATION AND METRICS ===
# Load teacher tokenizer and tokenize function
teacher_tokenizer = AutoTokenizer.from_pretrained(teacher_ckpt)
def tokenize_text(batch): return teacher_tokenizer(batch["text"], truncation=True)
# Tokenize datasets for teacher
clinc_enc = clinc.map(tokenize_text, batched=True, remove_columns=["text"])
clinc_enc = clinc_enc.rename_column("intent", "labels")

# Load student tokenizer and tokenize function
student_tokenizer = AutoTokenizer.from_pretrained(student_ckpt)
def tokenize_text_student(batch): return student_tokenizer(batch["text"], truncation=True)
# Tokenize datasets for student
clinc_enc_student = clinc.map(tokenize_text_student, batched=True, remove_columns=["text"])
clinc_enc_student = clinc_enc_student.rename_column("intent", "labels")

# Define compute_metrics function
accuracy_score = evaluate.load("accuracy")
f1_metric = evaluate.load("f1")
def compute_metrics(pred):
        """Computes accuracy and macro F1 score."""
        predictions, labels = pred
        predictions = np.argmax(predictions, axis=1) # Convert logits to class IDs

        accuracy_result = accuracy_score.compute(predictions=predictions, references=labels)
        f1_result = f1_metric.compute(
                predictions=predictions,
                references=labels,
                average="macro" # <-- This calculates the MACRO F1 score
        )'
        return {
                "accuracy": accuracy_result["accuracy"],
                "macro_f1": f1_result["f1"], # Return 'f1' key from the compute result
        }
```

```
# === TASK 1: FINE-TUNE TEACHER MODEL (ModernBERT-large) ===
teacher_training_args = TrainingArguments(
        output_dir="/kaggle/working/ModernBERT-large-finetuned-clinc",
        eval_strategy="epoch",
        num_train_epochs=5,
        learning_rate=2e-5,
        per_device_train_batch_size=16,
        per_device_eval_batch_size=16,
        weight_decay=0.01,
        logging_steps=100,
        save_strategy="epoch",
        load_best_model_at_end=True,
        metric_for_best_model="accuracy",
        greater_is_better=True,
        report_to="none",
        save_only_model=True,
        save_safetensors=True,
)

def model_init():
        """Initializes the teacher model."""
```

```
        return  AutoModelForSequenceClassification.from_pretrained(
                teacher_ckpt,  num_labels=num_labels,  id2label=id2label,  label2id=label2id
        )

teacher_trainer  =  Trainer(
        model_init=model_init,
        args=teacher_training_args,
        train_dataset=clinc_enc['train'],
        eval_dataset=clinc_enc['validation'],
        compute_metrics=compute_metrics,
        tokenizer=teacher_tokenizer,
)
#  Train  the  teacher  model
teacher_trainer.train()
#  Save  the  fine-tuned  teacher  model
save_path  =  "/kaggle/working/ModernBERT-large-finetuned-clinc"
teacher_trainer.save_model(save_path)
teacher_tokenizer.save_pretrained(save_path)
#  Upload  to  hugging  face
```

```
/tmp/ipython-input-1078479289.py:25: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Us
  teacher_trainer = Trainer(
config.json:         1.19k/? [00:00<00:00, 78.5kB/s]
model.safetensors: 100%                                                  1.58G/1.58G [00:41<00:00, 35.9MB/s]
Some weights of ModernBertForSequenceClassification were not initialized from the model checkpoint at answerdotai/ModernBERT-large and are ne
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
The tokenizer has new PAD/BOS/EOS tokens that differ from the model config and generation config. The model config and generation config were
Some weights of ModernBertForSequenceClassification were not initialized from the model checkpoint at answerdotai/ModernBERT-large and are ne
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
W1007 18:16:04.418000 1380 torch/_inductor/utils.py:1436] [1/0_1] Not enough SMs to use max_autotune_gemm mode
[4770/4770 28:31, Epoch 5/5]
```

Epoch	Training Loss	Validation Loss	Accuracy
1	0.168600	0.218951	0.952258
2	0.038900	0.185589	0.967742
3	0.018800	0.167427	0.969032
4	0.003300	0.165662	0.970323
5	0.000100	0.164647	0.969677

```
('/kaggle/working/ModernBERT-large-finetuned-clinc/tokenizer_config.json',
 '/kaggle/working/ModernBERT-large-finetuned-clinc/special_tokens_map.json',
 '/kaggle/working/ModernBERT-large-finetuned-clinc/tokenizer.json')
```

```
#  ===  TASK  2  &  3:  SHOW  RESULTS  AND  TEST  QUERY  ===
#  Task  2:  Show  main  training  results  (on  test  set)
def  tokenize_with_labels(batch):
        tokens  =  teacher_tokenizer(batch["text"],  truncation=True,  padding="max_length",  max_length=128)
        tokens["labels"]  =  batch["intent"]
        return  tokens
clinc_enc_test  =  clinc["test"].map(tokenize_with_labels,  batched=True)
trainer_metrics  =  teacher_trainer.evaluate(eval_dataset=clinc_enc_test)
print(trainer_metrics)
#  Upload  to  hugging  face
```

```
Map: 100%                                                  5500/5500 [00:02<00:00, 2809.07 examples/s]
[344/344 27:03]
{'eval_loss': 0.5416863560676575, 'eval_accuracy': 0.9003636363636364, 'eval_runtime': 139.9903, 'eval_samples_per_second': 39.288, 'eval_ste
```

```
#  Task  3:  Test  the  specific  query
pipe_teacher  =  pipeline(
        "text-classification",
        model="pine-cone/ModernBERT-large-finetuned-clinc",
        tokenizer="pine-cone/ModernBERT-large-finetuned-clinc",
)
query  =  "Hey,  I'd  like  to  rent  a  vehicle  from  Nov  1st  to  Nov  15th  in  Paris  and  I  need  a  15  passenger  van"
print(pipe_teacher(query))
```

```
config.json:          9.17k/? [00:00<00:00, 1.01MB/s]
model.safetensors: 100%                                                 1.58G/1.58G [01:23<00:00, 50.8MB/s]
tokenizer_config.json:          20.8k/? [00:00<00:00, 2.14MB/s]
tokenizer.json:          3.58M/? [00:00<00:00, 11.3MB/s]
special_tokens_map.json: 100%                                           694/694 [00:00<00:00, 82.2kB/s]
Device set to use cuda:0
[{'label': 'car_rental', 'score': 0.9993919134140015}]
```

```
# === TASK 4: KNOWLEDGE DISTILLATION ===

# Define custom Distillation Trainer
class DistillationTrainingArguments(TrainingArguments):
    def __init__(self, *args, alpha=0.5, temperature=2.0, **kwargs):
        super().__init__(*args, **kwargs)
        self.alpha, self.temperature = alpha, temperature
class DistillationTrainer(Trainer):
    def __init__(self, *args, teacher_model=None, **kwargs):
        super().__init__(*args, **kwargs)
        self.teacher_model = teacher_model
        if self.teacher_model:
            self.teacher_model.to(self.args.device); self.teacher_model.eval()
    def compute_loss(self, model, inputs, return_outputs=False, **kwargs):
        """Computes the distillation loss (KD + CE)."""

        # Get student logits and standard cross-entropy loss
        outputs_stu = model(**inputs);
        loss_ce, logits_stu = outputs_stu.loss, outputs_stu.logits

        # Get teacher logits (in no_grad context)
        with torch.no_grad(): logits_tea = self.teacher_model(**inputs).logits

        # Calculate Kullback-Leibler divergence loss
        loss_fct = nn.KLDivLoss(reduction="batchmean")
        loss_kd = self.args.temperature**2 * loss_fct(
            F.log_softmax(logits_stu/self.args.temperature, dim=-1),
            F.softmax(logits_tea/self.args.temperature, dim=-1)
        )

        # Combine losses
        loss = self.args.alpha*loss_ce + (1.-self.args.alpha)*loss_kd
        return (loss, outputs_stu) if return_outputs else loss
```

```
# Load the fine-tuned teacher model for distillation
teacher_model = AutoModelForSequenceClassification.from_pretrained("pine-cone/ModernBERT-large-finetuned-clinc")
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
teacher_model.to(device)
teacher_model.eval()

# Define student model initializer
def model_init():
    model = AutoModelForSequenceClassification.from_pretrained(
        student_ckpt,
        num_labels=teacher_model.config.num_labels
    )
    # Align labels with teacher model
    model.config.id2label = teacher_model.config.id2label
    model.config.label2id = teacher_model.config.label2id
    return model

# --- (2) Distilled Version (Fixed Params) ---
fixed_args = DistillationTrainingArguments(
    output_dir="/kaggle/working/bert-base-distilled-fixed",
    eval_strategy="epoch",
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    learning_rate=2e-5,
    num_train_epochs=4,
    weight_decay=0.01,
    save_strategy="epoch",
    load_best_model_at_end=True,
    alpha=0.5,
    temperature=7.0,
    report_to="none",
    save_only_model=True,
    logging_strategy="steps",
    logging_steps=10,
    gradient_accumulation_steps=2
```

```
)

fixed_trainer = DistillationTrainer(
        model_init=model_init,
        teacher_model=teacher_model,
        args=fixed_args,
        train_dataset=clinc_enc_student["train"],
        eval_dataset=clinc_enc_student["validation"],
        compute_metrics=compute_metrics,
        tokenizer=student_tokenizer,
)
fixed_trainer.train()
# Upload to hugging face
```

```
config.json:          9.17k/? [00:00<00:00, 779kB/s]
model.safetensors: 100%                                  1.58G/1.58G [01:00<00:00, 71.2MB/s]
/tmp/ipython-input-1428946420.py:8: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `DistillationTrainer.__
  super().__init__(*args, **kwargs)
model.safetensors: 100%                                  440M/440M [00:17<00:00, 17.5MB/s]
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized:
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized:
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
[1908/1908 12:13, Epoch 4/4]
```

Epoch	Training Loss	Validation Loss	Accuracy
1	2.710600	2.612303	0.647742
2	2.089700	2.003171	0.887419
3	1.757900	1.719881	0.924839
4	1.680700	1.636488	0.934839

```
TrainOutput(global_step=1908, training_loss=2.275734621023982, metrics={'train_runtime': 733.9495, 'train_samples_per_second': 83.112,
'train_steps_per_second': 2.6, 'total_flos': 575587515554556.0, 'train_loss': 2.275734621023982, 'epoch': 4.0})
```

```
# --- (3) Distilled Version (Optuna) ---
def hp_space(trial):
        return {
                """Defines the hyperparameter search space for Optuna."""
                "learning_rate": 2e-5,
                "num_train_epochs": 8,
                "alpha": trial.suggest_float("alpha", 0.4, 0.8),
                "temperature": trial.suggest_int("temperature", 2.0, 12.0)
        }

# Base args for Optuna search
distil_args = DistillationTrainingArguments(
        output_dir="/kaggle/working/bert-base-distilled-Optuna",
        eval_strategy="epoch",
        per_device_train_batch_size=16,
        per_device_eval_batch_size=16,
        learning_rate=2e-5,               # set by Optuna
        num_train_epochs=6,               # set by Optuna
        weight_decay=0.01,
        save_strategy="epoch",
        load_best_model_at_end=True,
        metric_for_best_model="accuracy",
        greater_is_better=True,
        alpha=0.5,                         # set by Optuna
        temperature=2.0,                   # set by Optuna
        report_to="none",
        save_only_model=True,
        logging_strategy="steps",
        logging_steps=10,
        gradient_accumulation_steps=2
)

search_trainer = DistillationTrainer(
        model_init=model_init,
        teacher_model=teacher_model,
        args=distil_args,
        train_dataset=clinc_enc_student["train"],
        eval_dataset=clinc_enc_student["validation"],
        compute_metrics=compute_metrics,
        tokenizer=student_tokenizer,
)
# Run HPO search
best_run = search_trainer.hyperparameter_search(n_trials=10, direction="maximize", hp_space=hp_space)
```

```
# Train final model with best Optuna parameters
best_params = best_run.hyperparameters

best_args = DistillationTrainingArguments(
        output_dir="/kaggle/working/bert-base-distilled-final",
        eval_strategy="epoch",
        per_device_train_batch_size=16,
        per_device_eval_batch_size=16,
        learning_rate=2e-5,
        num_train_epochs=8,
        weight_decay=0.01,
        warmup_ratio=0.06,
        save_strategy="epoch",
        load_best_model_at_end=True,
        metric_for_best_model="eval_macro_f1",
        greater_is_better=True,
        report_to="none",
        logging_strategy="steps",
        logging_steps=10,
        gradient_accumulation_steps=2,
        alpha=best_params["alpha"],
        temperature=best_params["temperature"]
)
best_model = model_init()
final_trainer = DistillationTrainer(
        model=best_model,
        teacher_model=teacher_model,
        args=best_args,
        train_dataset=clinc_enc_student["train"],
        eval_dataset=clinc_enc_student["validation"],
        compute_metrics=compute_metrics,
        tokenizer=student_tokenizer
)
final_trainer.train()
# Upload to hugging face
```

```
model.safetensors: 100%                                   440M/440M [00:05<00:00, 101MB/s]
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized:
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
/tmp/ipython-input-1428946420.py:8: FutureWarning: `tokenizer` is deprecated and will be removed in version 5.0.0 for `DistillationTrainer.__
  super().__init__(*args, **kwargs)
W1008 12:35:31.063000 1475 torch/_inductor/utils.py:1436] [1/0_1] Not enough SMs to use max_autotune_gemm mode
[3816/3816 26:35, Epoch 8/8]
```

Epoch	Training Loss	Validation Loss	Accuracy	Macro F1
1	2.987700	2.848179	0.760968	0.749750
2	1.685000	1.608866	0.914194	0.917152
3	1.122100	1.092647	0.943871	0.948558
4	0.862700	0.886773	0.950645	0.955031
5	0.718800	0.798898	0.953548	0.958233
6	0.641000	0.758899	0.954194	0.958709
7	0.632800	0.737825	0.955161	0.959764
8	0.626700	0.733441	0.953871	0.958537

```
TrainOutput(global_step=3816, training_loss=1.3709464952880732, metrics={'train_runtime': 1605.7092, 'train_samples_per_second': 75.979,
'train_steps_per_second': 2.377, 'total_flos': 1149331818924300.0, 'train_loss': 1.3709464952880732, 'epoch': 8.0})
```

```
# === TASK 5 & 6: BENCHMARKING AND PLOTTING ===
class PerformanceBenchmark:
        """Class to benchmark accuracy, latency, and model size."""
        def __init__(self, pipeline, dataset, optim_type):
                self.pipeline, self.dataset, self.optim_type = pipeline, dataset, optim_type
        def compute_accuracy(self):
                """Computes accuracy on the test set."""
                preds, labels = [], []
                for example in self.dataset:
                        pred = self.pipeline(example["text"])[0]["label"]
                        pred_label = pred.replace("LABEL_", "")
                        pred_idx = int(pred_label) if pred_label.isdigit() else intents.str2int(pred_label)
                        preds.append(pred_idx)
                        labels.append(example["intent"])
                return accuracy_score.compute(predictions=preds, references=labels)
        def compute_size(self):
                """Computes model size in MB."""
                state_dict = self.pipeline.model.state_dict()
                tmp_path = Path("model.pt"); torch.save(state_dict, tmp_path)
```

```
                size_mb = Path(tmp_path).stat().st_size/(1024*1024); tmp_path.unlink()
                return {"size_mb": size_mb}
        def time_pipeline(self, query="What is the pin number for my account?"):
                """Computes average latency in ms."""
                for _ in range(10): _ = self.pipeline(query)
                latencies = []
                for _ in range(100):
                        start = perf_counter(); _ = self.pipeline(query); latencies.append(perf_counter()-start)
                return {"time_avg_ms": 1000*np.mean(latencies), "time_std_ms": 1000*np.std(latencies)}
        def run_benchmark(self):
                """Runs all benchmarks."""
                m = {}; m[self.optim_type] = {}
                m[self.optim_type].update(self.compute_size())
                m[self.optim_type].update(self.time_pipeline())
                m[self.optim_type].update(self.compute_accuracy())
                return m

# Run benchmarks for all 4 models on CPU
perf_metrics = {}
clinc_test = load_dataset("clinc_oos", "plus", split="test")

# (1) ModernBERT-large (Baseline)
model_path = "pine-cone/ModernBERT-large-finetuned-clinc"
model = AutoModelForSequenceClassification.from_pretrained(model_path)
model.eval()
pipe_teacher_cpu = pipeline(
        "text-classification",
        model=model,
        tokenizer=model_path,
        device=-1    # CPU
)
pb = PerformanceBenchmark(pipe_teacher_cpu, clinc["test"], "ModernBERT-large")
perf_metrics.update(pb.run_benchmark())
```

```
# (2) Distillation-Fixed
model_path = "pine-cone/bert-base-distilled-fixed"
model = AutoModelForSequenceClassification.from_pretrained(model_path)
model.eval()
pipe_fixed_cpu = pipeline(
        "text-classification",
        model=model,
        tokenizer=model_path,
        device=-1
)
pb = PerformanceBenchmark(pipe_fixed_cpu, clinc["test"], "Distillation-Fixed")
perf_metrics.update(pb.run_benchmark())
```

```
# (3) Distillation-Optuna
model_path = "pine-cone/bert-base-distilled-final"
model = AutoModelForSequenceClassification.from_pretrained(model_path)
model.eval()
pipe_distilled_cpu = pipeline(
        "text-classification",
        model=model,
        tokenizer=model_path,
        device=-1
)
pb = PerformanceBenchmark(pipe_distilled_cpu, clinc["test"], "Distillation-Optuna")
perf_metrics.update(pb.run_benchmark())
```

```
# (4) Quantized-Distilled Version
model_name = "pine-cone/bert-base-distilled-Optuna"
output_path = "/kaggle/working/bert-base-distilled-final"
model_distilled = AutoModelForSequenceClassification.from_pretrained(output_path).to("cpu")
model_distilled.eval()
model_quantized = quantize_dynamic(model_distilled, {torch.nn.Linear}, dtype=torch.qint8)
pipe_quantized_cpu = pipeline(
        "text-classification",
        model=model_quantized,
        tokenizer=output_path,
        device=-1
)
model.eval()
pb = PerformanceBenchmark(pipe_quantized_cpu, clinc["test"], "Distillation+Quantization")
perf_metrics.update(pb.run_benchmark())
```

```
# Display final results table
final_df = pd.DataFrame.from_dict(perf_metrics, orient='index')
print(final_df)
```

	size_mb	time_avg_ms	time_std_ms	accuracy
ModernBERT-large	1510.624507	521.414343	137.759554	0.915091
Distillation-Fixed	418.147475	116.434022	9.250822	0.902182
Distillation-Optuna	418.147475	117.118999	8.689499	0.907818
Distillation+Quantization	173.182982	30.170859	2.525748	0.904182

```
# Plotting function (Task 6)
def plot_metrics(perf_metrics, current_optim_type=None):
    """Plots Accuracy vs Latency, with bubble size as model size."""
    df = pd.DataFrame.from_dict(perf_metrics, orient='index')

    for idx in df.index:
        df_opt = df.loc[idx]
        if idx == current_optim_type:
            plt.scatter(df_opt["time_avg_ms"], df_opt["accuracy"] * 100,
                        alpha=0.5, s=df_opt["size_mb"], label=idx,
                        marker='$\u25CC$')
        else:
            plt.scatter(df_opt["time_avg_ms"], df_opt["accuracy"] * 100,
                        s=df_opt["size_mb"], label=idx, alpha=0.5)

    plt.legend(bbox_to_anchor=(1,1))
    plt.ylabel("Accuracy (%)")
    plt.xlabel("Average Latency (ms)")
    plt.title("Accuracy vs Latency (bubble size = model size)")
    plt.ylim(85, 95)
    plt.grid(True)
    plt.show()

plot_metrics(perf_metrics, current_optim_type="DistilBERT")
```

{'ModernBERT-large': {'size_mb': 1510.6245069503784, 'time_avg_ms': np.float64(521.4143433500021), 'time_std_ms': np.float64(137.759553569851