MMAI 894 — Applied AI Project

Intent Classification with Fine-Tuned Transformer

Model: DistilRoBERTa + LoRA

Team: Museum

```
# ============================================================
# FIX VERSION CONFLICTS — PIN COMPATIBLE VERSIONS
# ============================================================
!pip install -q protobuf==3.20.*
!pip install -q --no-deps transformers==4.41.2 tokenizers==0.19.1
accelerate==0.33.0
!pip install -q --no-deps peft==0.11.1 datasets==2.21.0
!pip install -q scikit-learn==1.5.1
!pip install -q ftfy faker

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 162.1/162.1 kB 3.6 MB/s eta
0:00:00a 0:00:01
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
bigframes 2.12.0 requires google-cloud-bigquery-
storage<3.0.0,>=2.30.0, which is not installed.
opentelemetry-proto 1.37.0 requires protobuf<7.0,>=5.0, but you have
protobuf 3.20.3 which is incompatible.
onnx 1.18.0 requires protobuf>=4.25.1, but you have protobuf 3.20.3
which is incompatible.
a2a-sdk 0.3.10 requires protobuf>=5.29.5, but you have protobuf 3.20.3
which is incompatible.
ray 2.51.1 requires click!=8.3.0,>=7.0, but you have click 8.3.0 which
is incompatible.
bigframes 2.12.0 requires rich<14,>=12.4.4, but you have rich 14.2.0
which is incompatible.
tensorflow-metadata 1.17.2 requires protobuf>=4.25.2; python_version
>= "3.11", but you have protobuf 3.20.3 which is incompatible.
pydrive2 1.21.3 requires cryptography<44, but you have cryptography
46.0.3 which is incompatible.
pydrive2 1.21.3 requires pyOpenSSL<=24.2.1,>=19.1.0, but you have
pyopenssl 25.3.0 which is incompatible.
ydf 0.13.0 requires protobuf<7.0.0,>=5.29.1, but you have protobuf
3.20.3 which is incompatible.
grpcio-status 1.71.2 requires protobuf<6.0dev,>=5.26.1, but you have
protobuf 3.20.3 which is incompatible.
gcsfs 2025.3.0 requires fsspec==2025.3.0, but you have fsspec
```

```
2025.10.0 which is incompatible.
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 43.8/43.8 kB 1.7 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.1/9.1 MB 66.5 MB/s eta
0:00:0000:0100:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.6/3.6 MB 93.7 MB/s eta
0:00:00:00:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 315.1/315.1 kB 20.1 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 251.6/251.6 kB 5.5 MB/s eta
0:00:0000:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 527.3/527.3 kB 18.1 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.3/13.3 MB 9.3 MB/s eta
0:00:0000:0100:010m
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
cesium 0.12.4 requires numpy<3.0,>=2.0, but you have numpy 1.26.4
which is incompatible.
umap-learn 0.5.9.post2 requires scikit-learn>=1.6, but you have
scikit-learn 1.5.1 which is incompatible.
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 44.8/44.8 kB 997.2 kB/s eta
0:00:000:00:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 14.7 MB/s eta
0:00:00a 0:00:01
```

```python
# ============================================================
# IMPORTS
# ============================================================
import os, re, random, time
import numpy as np
import pandas as pd
import ftfy
from faker import Faker
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score

# Torch & HF Datasets
import torch
from datasets import Dataset

# HuggingFace Transformers
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    DataCollatorWithPadding,
    TrainingArguments,
    Trainer,
    EarlyStoppingCallback,
```

```
    MarianMTModel,
    MarianTokenizer,
)

# PEFT (LoRA)
from peft import (
    LoraConfig,
    get_peft_model,
    TaskType,
    PeftModel,
    PeftConfig
)
```

```
2025-11-22 18:53:03.784957: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to
register cuFFT factory: Attempting to register factory for plugin
cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are
written to STDERR
E0000 00:00:1763837583.988269      48 cuda_dnn.cc:8310] Unable to
register cuDNN factory: Attempting to register factory for plugin
cuDNN when one has already been registered
E0000 00:00:1763837584.044697      48 cuda_blas.cc:1418] Unable to
register cuBLAS factory: Attempting to register factory for plugin
cuBLAS when one has already been registered
```

SECTION 1 — Data Preparation Pipeline

Cleaning • Placeholder Augmentation • Category Tagging • Train/Val/Test Split

```
# ============================================================
# STEP 1 — LOAD DATASET
# ============================================================
ROOT = "../kaggle/" if "KAGGLE_KERNEL_RUN_TYPE" not in os.environ else
"/kaggle/"

INPUT_DIR = ROOT + "input/bitext-gen-ai-chatbot-customer-support-
dataset"
OUTPUT_DIR = ROOT + "working"
FILE_NAME =
"Bitext_Sample_Customer_Support_Training_Dataset_27K_responses-
v11.csv"

SEED = 42
random.seed(SEED)
np.random.seed(SEED)
```

```
os.makedirs(OUTPUT_DIR, exist_ok=True)

df = pd.read_csv(os.path.join(INPUT_DIR, FILE_NAME))
df = df[["instruction", "category", "intent"]].copy()

print("Loaded:", df.shape)


# ============================================================
# STEP 2 — FIX MOJIBAKE (â€™ → ’ etc.)
# ============================================================
def fix_mojibake(text):
    try:
        return ftfy.fix_text(text)
    except:
        return text

df["instruction"] = df["instruction"].astype(str).apply(fix_mojibake)


# ============================================================
# STEP 3 — PLACEHOLDER REPLACEMENT
# ============================================================

PLACEHOLDER_RE = re.compile(r"\{\{(.*?)\}\}")

def generate_placeholder_value(placeholder: str, faker: Faker) -> str:
    """
    Generate a value for a given placeholder name using Faker.
    """
    ph = placeholder.strip()

    if ph == "Order Number":
        return f"ORD{faker.random_int(min=100000, max=999999)}"

    elif ph == "Account Type":
        return faker.random_element(elements=[
            "Checking Account", "Savings Account",
            "Credit Card", "Business Account"
        ])

    elif ph == "Person Name":
        return faker.name()

    elif ph == "Account Category":
        return faker.random_element(elements=[
            "Personal", "Business", "Premium", "Student"
        ])

    elif ph == "Currency Symbol":
        return faker.random_element(elements=["$", "€", "£"])
```

```
    elif ph == "Refund Amount":
        amount = faker.pyfloat(min_value=5, max_value=250,
right_digits=2)
        return f"{amount:.2f}"

    elif ph == "Delivery City":
        return faker.city()

    elif ph == "Delivery Country":
        return faker.country()

    elif ph == "Invoice Number":
        return f"INV{faker.random_int(min=10000, max=99999)}"

    return ph  # fallback


def fill_instruction_placeholders(
    df: pd.DataFrame,
    col: str = "instruction",
    base_seed: int = 42,
    locale: str = "en_US",
) -> pd.DataFrame:
    """
    Replace {{placeholder}} with concrete values using Faker.
    """
    out = df.copy()

    def process_row(idx, text: str) -> str:
        text = str(text)
        placeholders = PLACEHOLDER_RE.findall(text)
        if not placeholders:
            return text

        faker = Faker(locale)
        faker.seed_instance(base_seed + int(idx))

        row_map = {}
        for ph in placeholders:
            key = ph.strip()
            if key not in row_map:
                row_map[key] = generate_placeholder_value(key, faker)

        def repl(match):
            ph_raw = match.group(1).strip()
            return row_map.get(ph_raw, ph_raw)

        return PLACEHOLDER_RE.sub(repl, text)
```

```
    out[col] = [process_row(idx, val) for idx, val in
out[col].items()]
    return out


df = fill_instruction_placeholders(
    df,
    col="instruction",
    base_seed=123,
    locale="en_US",
)


# ============================================================
# STEP 4A — CLEAN MINOR GARBAGE (KEEP NATURAL ERRORS)
# ============================================================
def clean_garbage(text):
    text = re.sub(r"[]+", "", text)
    text = re.sub(r"\s+", " ", text).strip()
    return text

df["instruction"] = df["instruction"].apply(clean_garbage)


# ============================================================
# STEP 4B — ADD STRUCTURED CATEGORY TAG FOR FT + LLM
# ============================================================
df["instruction"] = df.apply(
    lambda row: f"[CATEGORY={row['category']}] " + row["instruction"],
    axis=1
)


# ============================================================
# STEP 4C — REMOVE ORIGINAL CATEGORY COLUMN
# ============================================================
df = df.drop(columns=["category"])


# ============================================================
# STEP 5 — SAMPLE 15 ROWS TO VERIFY OUTPUT
# ============================================================
print("\n=== Sample cleaned rows ===")
print(df.sample(15, random_state=SEED))


# ============================================================
# STEP 6 — SAVE CLEANED DATASET
# ============================================================
cleaned_path = os.path.join(OUTPUT_DIR, "cleaned_bitext.csv")
```

```python
df.to_csv(cleaned_path, index=False)
print("\nSaved cleaned dataset →", cleaned_path)


# ============================================================
# STEP 7 — TRAIN/VAL/TEST SPLIT (80/10/10)
# ============================================================
train_df, temp_df = train_test_split(
    df, test_size=0.2, random_state=SEED, stratify=df["intent"]
)

val_df, test_df = train_test_split(
    temp_df, test_size=0.5, random_state=SEED,
stratify=temp_df["intent"]
)

train_df.to_csv(os.path.join(OUTPUT_DIR, "train.csv"), index=False)
val_df.to_csv(os.path.join(OUTPUT_DIR, "val.csv"), index=False)
test_df.to_csv(os.path.join(OUTPUT_DIR, "test.csv"), index=False)

print("\nSaved train/val/test splits.")
print(train_df.shape, val_df.shape, test_df.shape)
```

```
Loaded: (26872, 3)

=== Sample cleaned rows ===
                                                 instruction  \
9329   [CATEGORY=CONTACT] I can't talk with a human a...
4160   [CATEGORY=INVOICE] I have got to locate hte bi...
18500  [CATEGORY=PAYMENT] I cannot pay, help me to in...
8840   [CATEGORY=CONTACT] I want help speaking to cus...
5098   [CATEGORY=PAYMENT] I try to see th accepted pa...
17250  [CATEGORY=SUBSCRIPTION] where to sign up to th...
3589   [CATEGORY=CANCEL] I'd like to see the withdrwa...
9043     [CATEGORY=CONTACT] I want to speak with someone
15800  [CATEGORY=INVOICE] can you help me getting bil...
4384   [CATEGORY=INVOICE] I don't know how to take a ...
11150  [CATEGORY=ACCOUNT] I don't know how to delete ...
6417   [CATEGORY=REFUND] help me check in what cases ...
4186   [CATEGORY=INVOICE] is it possible to locate my...
7301   [CATEGORY=FEEDBACK] i want help to file a cons...
8267   [CATEGORY=CONTACT] uhave a free number to call...

                         intent
9329         contact_human_agent
4160               check_invoice
18500              payment_issue
8840    contact_customer_service
5098       check_payment_methods
17250    newsletter_subscription
```

```
3589     check_cancellation_fee
9043        contact_human_agent
15800                get_invoice
4384               check_invoice
11150             delete_account
6417         check_refund_policy
4186               check_invoice
7301                   complaint
8267   contact_customer_service

Saved cleaned dataset → /kaggle/working/cleaned_bitext.csv

Saved train/val/test splits.
(21497, 2) (2687, 2) (2688, 2)
```

SECTION 2 — Fine-Tune DistilRoBERTa with LoRA

Dataset Loading • Label Encoding • Tokenization • LoRA Adapter • Training & Saving

```
# ============================================================
# STEP 1 — Load train/val splits
# ============================================================

print("Train shape:", train_df.shape)
print("Val shape:", val_df.shape)

# ============================================================
# STEP 2 — Build label mapping
# ============================================================
labels = sorted(train_df["intent"].unique())
label2id = {lbl: i for i, lbl in enumerate(labels)}
id2label = {i: lbl for lbl, i in label2id.items()}
num_labels = len(labels)
print("Num labels:", num_labels)


# ============================================================
# STEP 3 — Create HuggingFace Datasets
# ============================================================

train_df = train_df.reset_index(drop=True)
val_df   = val_df.reset_index(drop=True)

train_ds = Dataset.from_pandas(train_df[["instruction", "intent"]])
```

```
val_ds   = Dataset.from_pandas(val_df[["instruction", "intent"]])

train_ds = train_ds.rename_column("intent", "label")
val_ds   = val_ds.rename_column("intent", "label")

def encode_labels(example):
    return {"label": label2id[example["label"]]}

train_ds = train_ds.map(encode_labels)
val_ds   = val_ds.map(encode_labels)

# ============================================================
# STEP 4 — Tokenize and Encode Samples
# ============================================================

BASE_MODEL_NAME = "distilroberta-base"
tokenizer = AutoTokenizer.from_pretrained(BASE_MODEL_NAME)

def encode_batch(batch):
    return tokenizer(
        batch["instruction"],
        truncation=True,
        padding=False,
        max_length=128,
    )

train_ds = train_ds.map(encode_batch, batched=True)
val_ds   = val_ds.map(encode_batch, batched=True)

train_ds = train_ds.remove_columns(["instruction"])
val_ds   = val_ds.remove_columns(["instruction"])

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)


# ============================================================
# STEP 5 — Load Base Model and Apply LoRA Adapter
# ============================================================

model = AutoModelForSequenceClassification.from_pretrained(
    BASE_MODEL_NAME,
    num_labels=num_labels,
    id2label=id2label,
    label2id=label2id,
)

lora_config = LoraConfig(
    task_type=TaskType.SEQ_CLS,
    r=32,
    lora_alpha=64,
```

```
    lora_dropout=0.1,
    bias="none",
)

model = get_peft_model(model, lora_config)
model.print_trainable_parameters()


# ============================================================
# STEP 6 — Configure TrainingArguments and Trainer
# ============================================================

FT_DIR = ROOT + "working/distilroberta_lora_ft"

training_args = TrainingArguments(
    output_dir=FT_DIR,
    learning_rate=5e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=32,
    num_train_epochs=10,
    eval_strategy="epoch",
    save_strategy="epoch",
    logging_steps=50,
    load_best_model_at_end=True,
    metric_for_best_model="f1",
    report_to="none",
)

def compute_metrics(eval_pred):
    logits, labels_np = eval_pred
    preds = np.argmax(logits, axis=-1)
    acc = accuracy_score(labels_np, preds)
    macro_f1 = f1_score(labels_np, preds, average="macro")
    return {"accuracy": acc, "f1": macro_f1}


trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_ds,
    eval_dataset=val_ds,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
    callbacks=[
        EarlyStoppingCallback(
            early_stopping_patience=3,
            early_stopping_threshold=0.0001,
        )
    ],
```

```
)

# ============================================================
# STEP 6B — Measure Training Time and GPU Cost
# ============================================================

start_train = time.time()
trainer.train()
end_train = time.time()

train_seconds = end_train - start_train
train_hours   = train_seconds / 3600

print(f"\nTraining time: {train_seconds:.2f} seconds
({train_hours:.4f} hours)")

# === GPU cost estimation (Google Cloud T4: $0.35 / hour) ===
T4_PRICE = 0.35
gpu_cost = train_hours * T4_PRICE
print(f"Estimated GPU training cost on T4: ${gpu_cost:.4f} USD")


# ------------------------------------------------------------
# 7. Save fine-tuned LoRA model
# ------------------------------------------------------------
trainer.save_model(FT_DIR)
tokenizer.save_pretrained(FT_DIR)
print("Saved fine-tuned LoRA model to:", FT_DIR)
```

```
Train shape: (21497, 2)
Val shape: (2687, 2)
Num labels: 27
```

{"model_id":"bad74e6e4eb448a4a5dbc3c428a6070d","version_major":2,"version_minor":0}

{"model_id":"00e43fa958dc42009e32f69489cf2ae7","version_major":2,"version_minor":0}

{"model_id":"c6fa54adf87f4b26b80825412dd64718","version_major":2,"version_minor":0}

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/
file_download.py:942: FutureWarning: `resume_download` is deprecated
and will be removed in version 1.0.0. Downloads always resume when
possible. If you want to force a new download, use
`force_download=True`.
  warnings.warn(
```

{"model_id":"39e69fec8a744e2eb5be4830196a1ab4","version_major":2,"version_minor":0}

{"model_id":"52c144200a4a4e6cbc082fae69ed37fb","version_major":2,"version_minor":0}

{"model_id":"25ffc02a7f6d4e1e86b38d7d7f329752","version_major":2,"version_minor":0}

{"model_id":"f3cbd8120e2449598a10eb7010c5be60","version_major":2,"version_minor":0}

```
Parameter 'function'=<function encode_batch at 0x7befe42ce340> of the
transform datasets.arrow_dataset.Dataset._map_single couldn't be
hashed properly, a random hash was used instead. Make sure your
transforms and parameters are serializable with pickle or dill for the
dataset fingerprinting and caching to work. If you reuse this
transform, the caching mechanism will consider it to be different from
the previous calls and recompute everything. This warning is only
showed once. Subsequent hashing failures won't be showed.
```

{"model_id":"a516300cb3ab4d62aa5fbd32b3148232","version_major":2,"version_minor":0}

{"model_id":"8060a813576948b09991b93fef9d9e18","version_major":2,"version_minor":0}

{"model_id":"0ef78954fdcc4b22a63f3ea0f2dba408","version_major":2,"version_minor":0}

```
Some weights of RobertaForSequenceClassification were not initialized
from the model checkpoint at distilroberta-base and are newly
initialized: ['classifier.dense.bias', 'classifier.dense.weight',
'classifier.out_proj.bias', 'classifier.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.

trainable params: 1,201,179 || all params: 83,340,342 || trainable%:
1.4413

<IPython.core.display.HTML object>

/usr/local/lib/python3.11/dist-packages/huggingface_hub/
file_download.py:942: FutureWarning: `resume_download` is deprecated
and will be removed in version 1.0.0. Downloads always resume when
possible. If you want to force a new download, use
`force_download=True`.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/huggingface_hub/file_download.
py:942: FutureWarning: `resume_download` is deprecated and will be
removed in version 1.0.0. Downloads always resume when possible. If
you want to force a new download, use `force_download=True`.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/huggingface_hub/file_download.
```

```
py:942: FutureWarning: `resume_download` is deprecated and will be
removed in version 1.0.0. Downloads always resume when possible. If
you want to force a new download, use `force_download=True`.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/huggingface_hub/file_download.
py:942: FutureWarning: `resume_download` is deprecated and will be
removed in version 1.0.0. Downloads always resume when possible. If
you want to force a new download, use `force_download=True`.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/huggingface_hub/file_download.
py:942: FutureWarning: `resume_download` is deprecated and will be
removed in version 1.0.0. Downloads always resume when possible. If
you want to force a new download, use `force_download=True`.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/huggingface_hub/file_download.
py:942: FutureWarning: `resume_download` is deprecated and will be
removed in version 1.0.0. Downloads always resume when possible. If
you want to force a new download, use `force_download=True`.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/huggingface_hub/file_download.
py:942: FutureWarning: `resume_download` is deprecated and will be
removed in version 1.0.0. Downloads always resume when possible. If
you want to force a new download, use `force_download=True`.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/huggingface_hub/file_download.
py:942: FutureWarning: `resume_download` is deprecated and will be
removed in version 1.0.0. Downloads always resume when possible. If
you want to force a new download, use `force_download=True`.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/huggingface_hub/file_download.
py:942: FutureWarning: `resume_download` is deprecated and will be
removed in version 1.0.0. Downloads always resume when possible. If
you want to force a new download, use `force_download=True`.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/huggingface_hub/file_download.
py:942: FutureWarning: `resume_download` is deprecated and will be
removed in version 1.0.0. Downloads always resume when possible. If
you want to force a new download, use `force_download=True`.
  warnings.warn(


Training time: 353.06 seconds (0.0981 hours)
Estimated GPU training cost on T4: $0.0343 USD

/usr/local/lib/python3.11/dist-packages/huggingface_hub/
file_download.py:942: FutureWarning: `resume_download` is deprecated
and will be removed in version 1.0.0. Downloads always resume when
possible. If you want to force a new download, use
`force_download=True`.
  warnings.warn(
```

```
Saved fine-tuned LoRA model to: /kaggle/working/distilroberta_lora_ft
```

SECTION 2B — Full Fine-Tuning DistilRoBERTa (No LoRA)

Dataset Loading • Full Parameter Training • Early Stopping • Save Checkpoint

```
print("\n===== SECTION 2B — Full Fine-Tuning DistilRoBERTa =====")

BASE_MODEL_NAME = "distilroberta-base"

# Reload tokenizer (safe)
fullft_tokenizer = AutoTokenizer.from_pretrained(BASE_MODEL_NAME)

# Rebuild dataset for Full FT (avoid shared references)
train_ds_ff = Dataset.from_pandas(train_df[["instruction",
"intent"]]).rename_column("intent", "label")
val_ds_ff   = Dataset.from_pandas(val_df[["instruction",
"intent"]]).rename_column("intent", "label")

# Encode labels
train_ds_ff = train_ds_ff.map(encode_labels)
val_ds_ff   = val_ds_ff.map(encode_labels)

# Tokenization
train_ds_ff = train_ds_ff.map(encode_batch, batched=True)
val_ds_ff   = val_ds_ff.map(encode_batch, batched=True)

train_ds_ff = train_ds_ff.remove_columns(["instruction"])
val_ds_ff   = val_ds_ff.remove_columns(["instruction"])

data_collator_ff = DataCollatorWithPadding(tokenizer=fullft_tokenizer)

# ------------------------------------------------------------
# Load base DistilRoBERTa for FULL fine-tuning (ALL params trainable)
# ------------------------------------------------------------
fullft_model = AutoModelForSequenceClassification.from_pretrained(
    BASE_MODEL_NAME,
    num_labels=num_labels,
    id2label=id2label,
    label2id=label2id
)

FULLFT_DIR = ROOT + "working/distilroberta_full_ft"
```

```
training_args_full = TrainingArguments(
    output_dir=FULLFT_DIR,
    learning_rate=3e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=32,
    num_train_epochs=8,
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="f1",
    logging_steps=50,
    report_to="none",
)

trainer_full = Trainer(
    model=fullft_model,
    args=training_args_full,
    train_dataset=train_ds_ff,
    eval_dataset=val_ds_ff,
    tokenizer=fullft_tokenizer,
    data_collator=data_collator_ff,
    compute_metrics=compute_metrics,
    callbacks=[
        EarlyStoppingCallback(early_stopping_patience=2)
    ]
)

# ------------------------------------------------------------
# Train + measure training time + compute GPU cost
# ------------------------------------------------------------
start_train_ff = time.time()
trainer_full.train()
end_train_ff = time.time()

train_seconds_ff = end_train_ff - start_train_ff
train_hours_ff   = train_seconds_ff / 3600

print(f"\n[Full FT] Training time: {train_seconds_ff:.2f} sec
({train_hours_ff:.4f} hours)")

T4_PRICE = 0.35
gpu_cost_ff = train_hours_ff * T4_PRICE
print(f"[Full FT] Estimated GPU cost: ${gpu_cost_ff:.4f}")

# Save model
trainer_full.save_model(FULLFT_DIR)
fullft_tokenizer.save_pretrained(FULLFT_DIR)

print("\nSaved FULL Fine-Tuned model →", FULLFT_DIR)
```

```
===== SECTION 2B — Full Fine-Tuning DistilRoBERTa =====

{"model_id":"d2c2891e4f3445aaa89bc26bfb4d6229","version_major":2,"vers
ion_minor":0}

{"model_id":"d6dfe9ccb692487e9195584511b1bec8","version_major":2,"vers
ion_minor":0}

{"model_id":"84c5ee21e3f54fe4a1d2faf6ecf1c52a","version_major":2,"vers
ion_minor":0}

{"model_id":"3622dd36176e47658b87346574737950","version_major":2,"vers
ion_minor":0}

Some weights of RobertaForSequenceClassification were not initialized
from the model checkpoint at distilroberta-base and are newly
initialized: ['classifier.dense.bias', 'classifier.dense.weight',
'classifier.out_proj.bias', 'classifier.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.

<IPython.core.display.HTML object>


[Full FT] Training time: 680.53 sec (0.1890 hours)
[Full FT] Estimated GPU cost: $0.0662

Saved FULL Fine-Tuned model → /kaggle/working/distilroberta_full_ft
```

SECTION 3 — DistilRoBERTa-LoRA Inference Pipeline

Load Model • Prepare Validation Set • Build Classifier • Run Inference • Evaluate Performance

```
# ============================================================
# STEP 0 — Load DistilRoBERTa-LoRA model
# ============================================================


# Path or HF repo name for your DistilRoBERTa-LoRA model
MODEL_DIR = ROOT + "/working/distilroberta_lora_ft"

# ------------------------------------------------------------
# 1) Load LoRA config to know which base model to use
# ------------------------------------------------------------
```

```
peft_config = PeftConfig.from_pretrained(MODEL_DIR)
base_model_name = peft_config.base_model_name_or_path  # e.g.
"distilroberta-base"

# ------------------------------------------------------------
# 2) Define your label mapping (must match training!)
# ------------------------------------------------------------
labels = sorted(train_df["intent"].unique())
label2id = {lbl: i for i, lbl in enumerate(labels)}
id2label = {i: lbl for lbl, i in label2id.items()}
num_labels = len(labels)  # should be 27

# ------------------------------------------------------------
# 3) Load tokenizer from the *base* model
# ------------------------------------------------------------
tokenizer = AutoTokenizer.from_pretrained(base_model_name)

# ------------------------------------------------------------
# 4) Load base DistilRoBERTa model with correct num_labels
# ------------------------------------------------------------
base_model = AutoModelForSequenceClassification.from_pretrained(
    base_model_name,
    num_labels=num_labels,
    id2label=id2label,
    label2id=label2id,
)

# ------------------------------------------------------------
# 5) Load LoRA adapter weights on top of the base model
# ------------------------------------------------------------
model = PeftModel.from_pretrained(base_model, MODEL_DIR)

# Device setup (use GPU if available)
lora_device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
print("Using:", lora_device)

model.to(lora_device)
model.eval()


# ============================================================
# STEP 1 — Load validation set & sample 270 items
# (10 samples per intent as required by assignment)
# ============================================================

val_df = pd.read_csv(ROOT + "working/val.csv")

sample_df = (
    val_df.groupby("intent")
```

```
          .head(10)
          .reset_index(drop=True)
)

print("Sample size:", sample_df.shape)
sample_df.head()


# ============================================================
# STEP 2 — Prepare intent list for LLM prompt
# ============================================================

all_intents = sorted(val_df["intent"].unique())
intent_list_str = ", ".join(all_intents)
print("Loaded intents:", len(all_intents))

# Bullet-style list
available_intents_bulleted = "\n- " + "\n- ".join(all_intents)


# ============================================================
# STEP 3 — Define DistilRoBERTa-LoRA classifier
# ============================================================

def _id_to_label(idx: int) -> str:
    """
    Map prediction index back to the string label using
model.config.id2label.
    Works whether id2label is a dict or a list.
    """
    id2label = model.config.id2label
    if isinstance(id2label, dict):
        # keys may be "0", "1", ... or ints
        return id2label.get(str(idx), id2label.get(idx, str(idx)))
    return id2label[idx]


def classify_distilroberta_lora(instruction: str) -> str:
    """
    Classify a single instruction using the DistilRoBERTa-LoRA model.
    """
    encoded = tokenizer(
        instruction,
        truncation=True,
        padding="max_length",
        max_length=128,
        return_tensors="pt",
    )

    # Send tensors to the same device as the model
```

```
    encoded = {k: v.to(lora_device) for k, v in encoded.items()}

    with torch.no_grad():
        outputs = model(**encoded)
        logits = outputs.logits
        pred_id = int(torch.argmax(logits, dim=-1))

    return _id_to_label(pred_id)


# ============================================================
# STEP 4 — Run inference on all 270 samples using DistilRoBERTa-LoRA
# ============================================================

val_preds = []
start = time.time()

# Run inference on validation samples
for text in sample_df["instruction"]:
    val_preds.append(classify_distilroberta_lora(text))

end = time.time()
elapsed = end - start

# Compute metrics
acc = accuracy_score(sample_df["intent"], val_preds)
f1 = f1_score(sample_df["intent"], val_preds, average="macro")

# Timing breakdown
time_per_sample = elapsed / len(sample_df)
time_per_1000 = time_per_sample * 1000

print("\n===== VALIDATION RESULTS =====")
print("Accuracy:", round(acc, 4))
print("Macro-F1:", round(f1, 4))

print("\n===== INFERENCE TIME =====")
print("Total time:", round(elapsed, 2), "seconds")
print("Time per sample:", round(time_per_sample, 4), "seconds")
print("Time per 1000 samples:", round(time_per_1000, 2), "seconds")
```

```
Some weights of RobertaForSequenceClassification were not initialized
from the model checkpoint at distilroberta-base and are newly
initialized: ['classifier.dense.bias', 'classifier.dense.weight',
'classifier.out_proj.bias', 'classifier.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.

Using: cuda
Sample size: (270, 2)
Loaded intents: 27
```

```
===== VALIDATION RESULTS =====
Accuracy: 0.9926
Macro-F1: 0.9926

===== INFERENCE TIME =====
Total time: 1.82 seconds
Time per sample: 0.0067 seconds
Time per 1000 samples: 6.73 seconds
```

SECTION 3B — DistilRoBERTa Full FT Inference Pipeline

Load Model • Prepare Validation Set • Run Classifier • Evaluate Accuracy & Macro-F1

```python
print("\n===== SECTION 3B — Full FT Validation Pipeline =====")

# Load trained model
fullft_model = AutoModelForSequenceClassification.from_pretrained(
    FULLFT_DIR,
    id2label=id2label,
    label2id=label2id
)

fullft_model.to(lora_device)
fullft_model.eval()

def classify_fullft(text):
    enc = fullft_tokenizer(
        text,
        truncation=True,
        padding="max_length",
        max_length=128,
        return_tensors="pt"
    )
    enc = {k: v.to(lora_device) for k, v in enc.items()}
    with torch.no_grad():
        logits = fullft_model(**enc).logits
        pred_id = int(torch.argmax(logits, -1))
    return id2label[pred_id]

# Run validation on 270 samples
val_preds_ff = []
start = time.time()
```

```
for text in sample_df["instruction"]:
    val_preds_ff.append(classify_fullft(text))

end = time.time()
elapsed_ff = end - start

acc_ff = accuracy_score(sample_df["intent"], val_preds_ff)
f1_ff  = f1_score(sample_df["intent"], val_preds_ff, average="macro")

print("\n===== FULL FT VALIDATION RESULTS =====")
print("Accuracy:", round(acc_ff, 4))
print("Macro-F1:", round(f1_ff, 4))
print("Total inference time:", round(elapsed_ff, 2), "sec")
print("Per 1000 samples:", round(elapsed_ff / 270 * 1000, 2), "sec")
```

```
===== SECTION 3B — Full FT Validation Pipeline =====

===== FULL FT VALIDATION RESULTS =====
Accuracy: 0.9963
Macro-F1: 0.9963
Total inference time: 1.39 sec
Per 1000 samples: 5.14 sec
```

SECTION 4 — Test Set Preprocessing & Translation Pipeline

Load raw test.csv • Apply Cleaning Pipeline • Placeholder Filling • Translation • Category Tagging

```
# ============================================================
# STEP A — Load raw test.csv
# ============================================================

TEST_PATH = ROOT + "input/test-csv/test.csv"
test_raw = pd.read_csv(TEST_PATH)

print("Loaded raw test:", test_raw.shape)
test_raw.head()


# ============================================================
# STEP B — Apply EXACT SAME CLEANING PIPELINE as TRAIN
# ============================================================

# --- 1) Fix mojibake ---
```

```
test_raw["instruction"] = (
    test_raw["instruction"]
        .astype(str)
        .apply(fix_mojibake)
)

# --- 2) Replace placeholders ---
test_raw = fill_instruction_placeholders(
    test_raw,
    col="instruction",
    base_seed=123,      # same as training
    locale="en_US",     # same as training
)

# --- 3) Clean garbage ---
test_raw["instruction"] = test_raw["instruction"].apply(clean_garbage)

# --- 4) Translation (Helsinki mul → en) ---

translation_model_name = "Helsinki-NLP/opus-mt-mul-en"
mt_tokenizer = MarianTokenizer.from_pretrained(translation_model_name)

mt_device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")
print("Translation model using:", mt_device)

mt_model =
MarianMTModel.from_pretrained(translation_model_name).to(mt_device)

def translate_to_en(text: str) -> str:
    batch = mt_tokenizer([text], return_tensors="pt", truncation=True)
    batch = {k: v.to(mt_device) for k, v in batch.items()}  # SAME
DEVICE
    gen = mt_model.generate(**batch, max_new_tokens=128)
    return mt_tokenizer.decode(gen[0], skip_special_tokens=True)

test_raw["instruction"] =
test_raw["instruction"].apply(translate_to_en)


# --- 5) Add category tag (if missing, mark as UNKNOWN) ---
if "category" in test_raw.columns:
    test_raw["instruction"] = test_raw.apply(
        lambda row: f"[CATEGORY={row['category']}] " +
row["instruction"],
        axis=1,
    )
else:
    def add_tag_auto(t):
        m = re.search(r"\[CATEGORY=([A-Z_]+)\]", t)
```

```
        if m:
            return t
        return "[CATEGORY=UNKNOWN] " + t

    test_raw["instruction"] =
test_raw["instruction"].apply(add_tag_auto)


print("\n=== Cleaned test sample ===")
print(test_raw.head())
```

```
Loaded raw test: (270, 5)
```


```
{"model_id":"fc079f172b6347b7b5eb9e012a57c7a2","version_major":2,"vers
ion_minor":0}

{"model_id":"c38e963a400f4eb39ec71d3f84f0a928","version_major":2,"vers
ion_minor":0}

{"model_id":"f8e4f314f9ed4012b038475d52921737","version_major":2,"vers
ion_minor":0}

{"model_id":"1f6cb4543aa84369b057d67f9492bb8e","version_major":2,"vers
ion_minor":0}
```


```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/
file_download.py:942: FutureWarning: `resume_download` is deprecated
and will be removed in version 1.0.0. Downloads always resume when
possible. If you want to force a new download, use
`force_download=True`.
  warnings.warn(
```


```
{"model_id":"97d3186632be4fca8ec09d0ad26fe086","version_major":2,"vers
ion_minor":0}
```


```
/usr/local/lib/python3.11/dist-packages/transformers/models/marian/
tokenization_marian.py:175: UserWarning: Recommended: pip install
sacremoses.
  warnings.warn("Recommended: pip install sacremoses.")

Translation model using: cuda
```


```
{"model_id":"e639182a3d8644acba1a43d41c8f798a","version_major":2,"vers
ion_minor":0}

{"model_id":"8e35fc050d144dddbd1c4c9add98a852","version_major":2,"vers
ion_minor":0}
```


```
=== Cleaned test sample ===
   id flags                                        instruction
category  \
```

```
0   1    BQ  [CATEGORY=CONTACT] Could you please let me kno...
CONTACT
1   2    BQ  [CATEGORY=ACCOUNT] I'm trying to switch from m...
ACCOUNT
2   3     B  [CATEGORY=CONTACT] Can someone from customer s...
CONTACT
3   4    BQ  [CATEGORY=ACCOUNT] Could you guide me through ...
ACCOUNT
4   5    BQ  [CATEGORY=CONTACT] I've tried to solve my prob...
CONTACT

                                                 response
0  Thank you for contacting us. We will assist yo...
1  Thank you for contacting us. We will assist yo...
2  Thank you for contacting us. We will assist yo...
3  Thank you for contacting us. We will assist yo...
4  Thank you for contacting us. We will assist yo...
```

SECTION 5 — DistilRoBERTa-LoRA Test Set Inference & Submission

Load Classifier • Run Test Inference • Build Submission • Compute Inference Cost

```
# ============================================================
# STEP C0 — Load DistilRoBERTa-LoRA classifier for inference
# ============================================================

MODEL_DIR = ROOT + "/working/distilroberta_lora_ft"  # your LoRA
folder

# ------------------------------------------------------------
# 1) Read LoRA config to get base model name
# ------------------------------------------------------------
peft_config = PeftConfig.from_pretrained(MODEL_DIR)
base_model_name = peft_config.base_model_name_or_path  # e.g.
"distilroberta-base"
print("LoRA base model:", base_model_name)

# ------------------------------------------------------------
# 2) Recreate label mapping (must match training)
# ------------------------------------------------------------
INTENT_LABELS = [
    "cancel_order",
    "change_order",
    "change_shipping_address",
```

```
    "check_cancellation_fee",
    "check_invoice",
    "check_payment_methods",
    "check_refund_policy",
    "complaint",
    "contact_customer_service",
    "contact_human_agent",
    "create_account",
    "delete_account",
    "delivery_options",
    "delivery_period",
    "edit_account",
    "get_invoice",
    "get_refund",
    "newsletter_subscription",
    "payment_issue",
    "place_order",
    "recover_password",
    "registration_problems",
    "review",
    "set_up_shipping_address",
    "switch_account",
    "track_order",
    "track_refund",
]

label2id = {lbl: i for i, lbl in enumerate(INTENT_LABELS)}
id2label = {i: lbl for lbl, i in label2id.items()}

# ------------------------------------------------------------
# 3) Load tokenizer for the classifier
# ------------------------------------------------------------
clf_tokenizer = AutoTokenizer.from_pretrained(base_model_name)

# ------------------------------------------------------------
# 4) Load base DistilRoBERTa with correct num_labels
# ------------------------------------------------------------
base_clf = AutoModelForSequenceClassification.from_pretrained(
    base_model_name,
    num_labels=len(INTENT_LABELS),
    id2label=id2label,
    label2id=label2id,
)

# ------------------------------------------------------------
# 5) Load LoRA adapter weights
# ------------------------------------------------------------
clf_model = PeftModel.from_pretrained(base_clf, MODEL_DIR)

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

```
"cpu")
clf_model.to(clf_device)
clf_model.eval()

# ------------------------------------------------------------
# C0.5 — Report Model Size (Base + LoRA Trainable Parameters)
# ------------------------------------------------------------
def count_parameters(model):
    total = sum(p.numel() for p in model.parameters())
    trainable = sum(p.numel() for p in model.parameters() if
p.requires_grad)
    return total, trainable

base_total, base_trainable = count_parameters(base_clf)
print(f"\nBase Model Total Params: {base_total:,}")
print(f"Base Model Trainable Params (all frozen): {base_trainable:,}")

lora_total, lora_trainable = count_parameters(clf_model)
print(f"\nLoRA+Base Total Params: {lora_total:,}")
print(f"LoRA Trainable Params (adapter only): {lora_trainable:,}")

print("\n=== Model Size Report Completed ===\n")



# ------------------------------------------------------------
# Classification function
# ------------------------------------------------------------
def classify_distilroberta_lora(text: str) -> str:
    enc = clf_tokenizer(
        text,
        truncation=True,
        padding="max_length",
        max_length=128,
        return_tensors="pt",
    )
    enc = {k: v.to(clf_device) for k, v in enc.items()}

    with torch.no_grad():
        outputs = clf_model(**enc)
        logits = outputs.logits
        pred_id = int(torch.argmax(logits, dim=-1))

    return id2label[pred_id]

print("LoRA base model:", base_model_name)


# ============================================================
# STEP C — Run inference on test rows (DistilRoBERTa-LoRA)
```

```
# ============================================================


test_preds = []
start = time.time()

for i, row in test_raw.iterrows():
    msg = row["instruction"]
    pred = classify_distilroberta_lora(msg)
    test_preds.append(pred)

    if i % 20 == 0:
        print(f"Processed {i}/{len(test_raw)}")

end = time.time()
elapsed_test = end - start


# ============================================================
# STEP D — Build submission file
# ============================================================

submission = pd.DataFrame({
    "id": test_raw["id"],
    "intent": test_preds
})

sub_path = ROOT + "working/submission.csv"
submission.to_csv(sub_path, index=False)

print("\nSaved submission →", sub_path)
print(submission.head())


# ============================================================
# STEP E — Report inference speed + cost
# ============================================================

time_per_sample_test = elapsed_test / len(test_raw)
time_per_1000_test = time_per_sample_test * 1000

print("\n===== INFERENCE REPORT (TEST SET) =====")
print("Total inference time:", round(elapsed_test, 2), "seconds")
print("Estimated time per 1000 samples:", round(time_per_1000_test,
2), "seconds")

# GPU Cost Calculation (T4)
T4_PRICE_PER_HOUR = 0.35
infer_hours_per_1000 = time_per_1000_test / 3600
gpu_infer_cost_per_1000 = infer_hours_per_1000 * T4_PRICE_PER_HOUR
```

```
print(f"Estimated Inference GPU Cost per 1000 samples (T4): $
{gpu_infer_cost_per_1000:.6f}")
print("===== DONE =====")
```

```
LoRA base model: distilroberta-base

Some weights of RobertaForSequenceClassification were not initialized
from the model checkpoint at distilroberta-base and are newly
initialized: ['classifier.dense.bias', 'classifier.dense.weight',
'classifier.out_proj.bias', 'classifier.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.


Base Model Total Params: 83,340,342
Base Model Trainable Params (all frozen): 611,355

LoRA+Base Total Params: 83,340,342
LoRA Trainable Params (adapter only): 611,355

=== Model Size Report Completed ===

LoRA base model: distilroberta-base
Processed 0/270
Processed 20/270
Processed 40/270
Processed 60/270
Processed 80/270
Processed 100/270
Processed 120/270
Processed 140/270
Processed 160/270
Processed 180/270
Processed 200/270
Processed 220/270
Processed 240/270
Processed 260/270

Saved submission → /kaggle/working/submission.csv
   id                    intent
0   1  contact_customer_service
1   2           switch_account
2   3  contact_customer_service
3   4           create_account
4   5       contact_human_agent

===== INFERENCE REPORT (TEST SET) =====
Total inference time: 1.86 seconds
Estimated time per 1000 samples: 6.89 seconds
Estimated Inference GPU Cost per 1000 samples (T4): $0.000670
===== DONE =====
```

This output corresponds to the fine-tuned DistilRoBERTa (LoRA) run submitted on Kaggle for the FT leaderboard entry.

SECTION 5B — DistilRoBERTa Full FT Test Set Inference & Submission

Load Full-FT Model • Predict Test Rows • Build Submission • Compute Inference Cost

```
print("\n===== SECTION 5B — Full FT Test Inference & Submission
=====")

test_preds_ff = []
start = time.time()

for i, row in test_raw.iterrows():
    pred = classify_fullft(row["instruction"])
    test_preds_ff.append(pred)

end = time.time()
elapsed_test_ff = end - start

submission_ff = pd.DataFrame({
    "id": test_raw["id"],
    "intent": test_preds_ff
})

sub_path_ff = ROOT + "working/submission_fullft.csv"
submission_ff.to_csv(sub_path_ff, index=False)

print("\nSaved FULL FT submission:", sub_path_ff)

# Compute inference cost
time_per_sample_ff = elapsed_test_ff / len(test_raw)
time_per_1000_ff   = time_per_sample_ff * 1000

infer_hours_ff = time_per_1000_ff / 3600
gpu_infer_cost_ff = infer_hours_ff * T4_PRICE

print("\n===== FULL FT TEST INFERENCE REPORT =====")
print("Inference time:", round(elapsed_test_ff, 2), "sec")
print("Per 1000 samples:", round(time_per_1000_ff, 2), "sec")
print(f"Estimated GPU inference cost (per 1000): $
{gpu_infer_cost_ff:.6f}")
print("===== DONE (FULL FINE-TUNING) =====")
```

```
===== SECTION 5B — Full FT Test Inference & Submission =====

Saved FULL FT submission: /kaggle/working/submission_fullft.csv

===== FULL FT TEST INFERENCE REPORT =====
Inference time: 1.46 sec
Per 1000 samples: 5.41 sec
Estimated GPU inference cost (per 1000): $0.000526
===== DONE (FULL FINE-TUNING) =====
```