MMAI 894 — Applied AI Project

Intent Classification with Large Language Model

Model: OpenAI GPT-4.1-mini

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
!pip install -q langdetect sentencepiece
!pip install -q openai
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 162.1/162.1 kB 2.7 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 1.2 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.1/9.1 MB 54.2 MB/s eta
0:00:0000:0100:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.6/3.6 MB 82.6 MB/s eta
0:00:00:00:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 315.1/315.1 kB 16.5 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 251.6/251.6 kB 4.6 MB/s eta
0:00:00a 0:00:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 527.3/527.3 kB 13.6 MB/s eta
0:00:0000:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.3/13.3 MB 69.8 MB/s eta
0:00:00:00:010:01
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.
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 1.2 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.0/2.0 MB 20.5 MB/s eta
0:00:00a 0:00:01
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 981.5/981.5 kB 10.2 MB/s eta
0:00:00a 0:00:01
etadata (setup.py) ...

# ============================================================
# IMPORTS
# ============================================================
import os
import re
import random
import time
import numpy as np
import pandas as pd
import ftfy
from faker import Faker

# ===========================
# Machine Learning Tools
# ===========================
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score
```

```
# ===========================
# Language Detection
# ===========================
from langdetect import detect, DetectorFactory
DetectorFactory.seed = 42

# ===========================
# Transformers / HuggingFace
# ===========================
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
)

# ===========================
# OpenAI API
# ===========================
from kaggle_secrets import UserSecretsClient
from openai import OpenAI
```

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(
```

```python
    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")
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 — GPT-4.1-mini LLM Evaluation Pipeline

#### *Sampling • Prompt Construction • Inference • Accuracy • Macro-F1 • Cost & Time Estimation*

```python
# ============================================================
# STEP 0 Load API Keys
# ============================================================

# Load API key
user_secrets = UserSecretsClient()
OPENAI_API_KEY = user_secrets.get_secret("OPENAI_API_KEY")
client = OpenAI(api_key=OPENAI_API_KEY)


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

val_df = pd.read_csv("/kaggle/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))
```

```
# STEP 3 — Build bullet-style intent list (required!)
available_intents_bulleted = "\n- " + "\n- ".join(all_intents)



# ============================================================
# STEP 3B — Define GPT-4.1-mini classifier
# ============================================================
def classify_gpt41mini(instruction):
    prompt = f"""
You are an intent classification assistant for customer support
messages.
Your task is to identify the single best intent that matches what the
user is trying to accomplish.
You MUST output exactly one intent from the allowed list.

User messages often begin with:
[CATEGORY=XXX] <instruction>
CATEGORY indicates which group of intents the message belongs to.
You MUST pick an intent ONLY from the intents listed under that
CATEGORY. Never choose an intent from any other category.

Here are the valid categories and their allowed intents:

ACCOUNT:
- create_account
- delete_account
- edit_account
- recover_password
- registration_problems
- switch_account

CANCELLATION_FEE:
- check_cancellation_fee

CONTACT:
- contact_customer_service
- contact_human_agent

DELIVERY:
- delivery_options
- delivery_period

FEEDBACK:
- complaint
- review

INVOICE:
- check_invoice
- get_invoice

ORDER:
- cancel_order
- change_order
- place_order
- track_order

PAYMENT:
- check_payment_methods
- payment_issue

REFUND:
- check_refund_policy
- get_refund
- track_refund

SHIPPING_ADDRESS:
- change_shipping_address
- set_up_shipping_address

SUBSCRIPTION:
- newsletter_subscription

Soft guidelines (not strict rules):
- Refund messages may involve requesting a refund, checking status, or
asking about refund rules.
- Delivery questions may be about timing, availability, or tracking.
- Setting up a new address ≠ editing an existing address.
- Account problems may involve recovery, switching profiles, or
editing information.
- Asking for help does not automatically mean contact_human_agent.
- Invoice questions may involve locating an existing invoice or
requesting a copy.

After determining the best intent, output ONLY the intent label in
this exact format:
Final: <intent>

Do NOT output reasoning, sentences, bullet points, or any other text.

User message:
"{instruction}"

"""


    try:
        response = client.chat.completions.create(
            model="gpt-4.1-mini",
            messages=[{"role": "user", "content": prompt}],
            temperature=0,
            max_tokens=64
        )
        output = response.choices[0].message.content.strip()

        # === Extract the final intent ===
        match = re.search(r"Final:\s*([a-zA-Z0-9_]+)", output)
        if match:
            return match.group(1).strip()
        else:
            # fallback: try last line
            last_line = output.splitlines()[-1].strip()
            return last_line.replace("Final:", "").strip()

    except Exception as e:
        print("Error:", e)
        return "ERROR"


# ============================================================
# STEP 4 — Run inference on all 270 samples
# ============================================================

preds = []
start = time.time()

for i, row in sample_df.iterrows():
    msg = row["instruction"]
    pred = classify_gpt41mini(msg)
    preds.append(pred)

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

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


sample_df["gpt41mini_pred"] = preds

save_path = "/kaggle/working/gpt41mini_eval.csv"
sample_df.to_csv(save_path, index=False)
print("Predictions saved →", save_path)


# ============================================================
# STEP 5 — Compute Accuracy & Macro-F1
# ============================================================

y_true = sample_df["intent"]
```

```
y_pred = sample_df["gpt41mini_pred"]

acc = accuracy_score(y_true, y_pred)
macro_f1 = f1_score(y_true, y_pred, average="macro")

print("\n===== GPT-4.1-mini Evaluation =====")
print("Accuracy:", acc)
print("Macro-F1:", macro_f1)


# ============================================================
# STEP 6 — Estimate inference time per 1000 samples
# ============================================================

time_per_sample = elapsed / len(sample_df)
time_per_1000 = time_per_sample * 1000

print("\nInference time for 270 samples:", elapsed, "seconds")
print("Estimated inference time per 1000 samples:", time_per_1000,
"seconds")


# ============================================================
# STEP 7 — Estimate API cost (per assignment requirement)
# ============================================================

# GPT-4.1-mini pricing:
# Input: $0.06 per 1M tokens
# Output: $0.12 per 1M tokens

avg_input_tokens = 350   # prompt + instruction
avg_output_tokens = 3    # label only

cost_per_1000 = (
    (avg_input_tokens * 1000 / 1_000_000) * 0.06 +
    (avg_output_tokens * 1000 / 1_000_000) * 0.12
)

print("\nEstimated API cost per 1000 samples: $", round(cost_per_1000,
5))
print("\n===== DONE =====")
```

```
Sample size: (270, 2)
Loaded intents: 27
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
Predictions saved → /kaggle/working/gpt41mini_eval.csv

===== GPT-4.1-mini Evaluation =====
Accuracy: 0.9666666666666667
Macro-F1: 0.9663580950130658

Inference time for 270 samples: 173.5626242160797 seconds
Estimated inference time per 1000 samples: 642.8245341336286 seconds

Estimated API cost per 1000 samples: $ 0.02136

===== DONE =====
```

SECTION 3 — GPT-4.1-mini Test Set Inference & Submission Pipeline

#### *Cleaning • Placeholder Augmentation • Category Tagging • LLM Inference • Submission Build • Cost & Time Estimation*

`

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

TEST_PATH = "/kaggle/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 (USE THE SAME FUNCTION AS TRAIN) ---
test_raw = fill_instruction_placeholders(
    test_raw,
    col="instruction",
    base_seed=123,
    locale="en_US"
)

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

# --- 4) Add category tag ---
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)



# ============================================================
# STEP C — Run inference on test rows
# ============================================================

test_preds = []
start = time.time()

for i, row in test_raw.iterrows():
    msg = row["instruction"]
    pred = classify_gpt41mini(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 = "/kaggle/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")

# --- Same cost formula as validation ---
avg_input_tokens = 350
avg_output_tokens = 3

estimated_cost_per_1000 = (
    (avg_input_tokens * 1000 / 1_000_000) * 0.06 +
    (avg_output_tokens * 1000 / 1_000_000) * 0.12
)

print("Estimated API cost per 1000 samples: $",
round(estimated_cost_per_1000, 5))
print("===== DONE =====")
```

```
Loaded raw test: (270, 5)
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: 157.96 seconds
Estimated time per 1000 samples: 585.05 seconds
Estimated API cost per 1000 samples: $ 0.02136
===== DONE =====
```