In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import torch
#import torch.nn as nn
#import accelerate
from datasets import load_dataset

from sklearn.metrics import accuracy_score, f1_score
from collections import Counter

from transformers import BertTokenizer, BertForSequenceClassification
from transformers import Trainer, TrainingArguments, pipeline

from peft import LoraConfig, get_peft_model, TaskType

import os
os.environ["WANDB_DISABLED"] = "true"
```

In [2]:
```
# Check if GPU is available, becouse training on CPU is very slow!!!
print(torch.cuda.is_available())
print(torch.cuda.get_device_name(0) if torch.cuda.is_available() else "No GPU")
```

```
True
NVIDIA GeForce RTX 3050 Ti Laptop GPU
```

In [3]:
```
dataset = load_dataset("go_emotions")

emotions = dataset["train"].features["labels"].feature.names
num_labels = len(emotions)
print(len(dataset["train"]))
print(num_labels)
print(emotions)
```

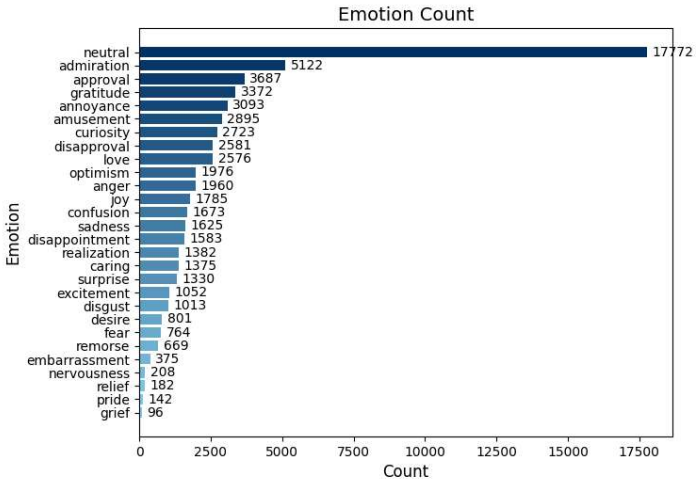
```
43410
28
['admiration', 'amusement', 'anger', 'annoyance', 'approval', 'caring', 'confusion',
'curiosity', 'desire', 'disappointment', 'disapproval', 'disgust', 'embarrassment',
'excitement', 'fear', 'gratitude', 'grief', 'joy', 'love', 'nervousness', 'optimis
m', 'pride', 'realization', 'relief', 'remorse', 'sadness', 'surprise', 'neutral']
```

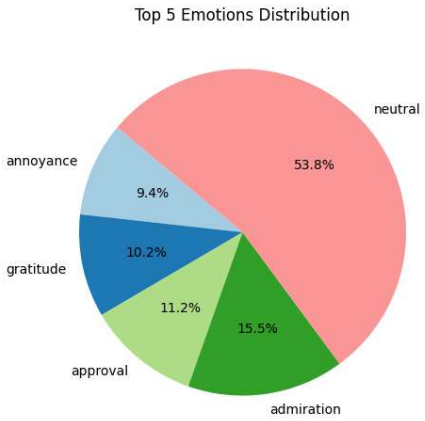
In [4]:
```
# Get the distribution of emotions in the dataset
emotion_counter = Counter()

for split, split_data in dataset.items():
    for sample in split_data:
        emotion_counter.update(sample["labels"])

emotions_count = {emotion: emotion_counter.get(i, 0) for i, emotion in enumerate(em

print(emotions_count)
```

```
{'admiration': 5122, 'amusement': 2895, 'anger': 1960, 'annoyance': 3093, 'approva
l': 3687, 'caring': 1375, 'confusion': 1673, 'curiosity': 2723, 'desire': 801, 'disa
ppointment': 1583, 'disapproval': 2581, 'disgust': 1013, 'embarrassment': 375, 'exci
tement': 1052, 'fear': 764, 'gratitude': 3372, 'grief': 96, 'joy': 1785, 'love': 257
6, 'nervousness': 208, 'optimism': 1976, 'pride': 142, 'realization': 1382, 'relie
f': 182, 'remorse': 669, 'sadness': 1625, 'surprise': 1330, 'neutral': 17772}
```

In [5]:

```
# --- Sort by ascending count ---
sorted_items = sorted(emotions_count.items(), key=lambda x: x[1])
emotions_sorted, counts_sorted = zip(*sorted_items)

fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# --- Horizontal bar plot (left) ---
import matplotlib.colors as mcolors

# Create a colormap from darkblue to lightblue
cmap = mcolors.LinearSegmentedColormap.from_list("blue_gradient", ["#87CEEB", "#003
# Generate colors from the colormap for each bar
colors = [cmap(i / (len(emotions_sorted) - 1)) for i in range(len(emotions_sorted))
bars = axes[0].barh(emotions_sorted, counts_sorted, color=colors)
for bar, count in zip(bars, counts_sorted):
    axes[0].text(
        bar.get_width() + max(counts_sorted) * 0.01,
        bar.get_y() + bar.get_height() / 2,
        str(count),
        va='center',
        ha='left',
        fontsize=10,
        fontweight='normal'
    )
axes[0].set_xlabel("Count", fontsize=12)
axes[0].set_ylabel("Emotion", fontsize=12)
axes[0].set_title("Emotion Count", fontsize=14)

# Show total train data at the bottom right
axes[0].text(
    1.0, -0.15, f"Total train samples: {len(dataset['train'])}",
    transform=axes[0].transAxes,
    fontsize=12,
    color="navy",
    ha="right",
    va="top",
    fontweight="bold"
)

# --- Pie chart of top 5 emotions (right) ---
top5_emotions = emotions_sorted[-5:]
top5_counts = counts_sorted[-5:]
axes[1].pie(top5_counts, labels=top5_emotions, autopct='%1.1f%%', startangle=140, c
axes[1].set_title("Top 5 Emotions Distribution")

plt.tight_layout(pad=2)
plt.show()
```





Total train samples: 43410

In [6]:
```
def multi_hot_encode(samples):
    multi_hot = np.zeros(len(emotions))

    for label in samples["labels"]:
        multi_hot[label] = 1

    return {"labels": multi_hot.tolist()}

dataset = dataset.map(multi_hot_encode)
```

In [7]:
```
model_name = 'bert-base-uncased'

tokenizer = BertTokenizer.from_pretrained(model_name)

def tokenize_function(examples):
    return tokenizer(examples["text"], padding="max_length", truncation=True, max_l

tokenized_dataset = dataset.map(tokenize_function, batched=True)

# Set the format of the dataset to PyTorch tensors
tokenized_dataset.set_format("torch")
```

In [8]:
```
convert_to_float = lambda sample: {"float_labels": sample["labels"].to(torch.float)

tokenized_dataset = (tokenized_dataset
                     .map(convert_to_float, remove_columns=["labels"])
                     .rename_column("float_labels", "labels"))
```

In [9]:
```
model = BertForSequenceClassification.from_pretrained(
    model_name,
    num_labels=len(emotions),
    problem_type="multi_label_classification"
)

# LoRA config
lora_config = LoraConfig(
    task_type=TaskType.SEQ_CLS,
    r=8,
```

```
    lora_alpha=16,
    lora_dropout=0.05,
    target_modules=["query", "key", "value"]
)

model = get_peft_model(model, lora_config)

# Show how many params are trainable now
model.print_trainable_parameters()
```

```
Some weights of BertForSequenceClassification were not initialized from the model ch
eckpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classi
fier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for
predictions and inference.
trainable params: 463,900 || all params: 109,967,672 || trainable%: 0.4219
```

In [10]:
```
device = "cuda:0" if torch.cuda.is_available() else "cpu"
print(device)
model = model.to(device)
```

```
cuda:0
```

In [11]:
```
# Define parameters
LEARNING_RATE = 5e-5
BATCH_SIZE = 16
EPOCHS = 10
WEIGHT_DECAY = 1e-2
```

In [12]:
```
training_args = TrainingArguments(
    output_dir="./results",
    eval_strategy="epoch",
    learning_rate=LEARNING_RATE,
    per_device_train_batch_size=BATCH_SIZE,
    per_device_eval_batch_size=BATCH_SIZE,
    num_train_epochs=EPOCHS,
    weight_decay=WEIGHT_DECAY,
    fp16=True
)
```

```
Using the `WANDB_DISABLED` environment variable is deprecated and will be removed in
v5. Use the --report_to flag to control the integrations used for logging result (fo
r instance --report_to none).
```

In [13]:
```
def compute_metrics(pred):
    logits, labels = pred
    preds = (logits > 0.5).astype(int) # Convert probabilities to binary prediction

    f1 = f1_score(labels, preds, average='micro')
    acc = accuracy_score(labels, preds)

    return {
        'f1': f1,
        'accuracy': acc
    }
```

In [14]:
```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset["train"],
    eval_dataset=tokenized_dataset["validation"],
    compute_metrics=compute_metrics
)
```

In [15]:
```
trainer.train()
```

[27140/27140 55:28, Epoch 10/10]

Epoch	Training Loss	Validation Loss	F1	Accuracy
1	0.141100	0.134053	0.029960	0.016771
2	0.121100	0.114641	0.228403	0.136749
3	0.108400	0.105937	0.318577	0.200700
4	0.105300	0.101055	0.364821	0.239219
5	0.100500	0.098409	0.392921	0.258201
6	0.097700	0.096625	0.422833	0.286215
7	0.097400	0.095831	0.419112	0.280686
8	0.095900	0.094932	0.431758	0.293034
9	0.097000	0.094348	0.426759	0.287505
10	0.095400	0.094235	0.428966	0.288979

Out[15]:
```
TrainOutput(global_step=27140, training_loss=0.10907475394893412, metrics={'train_
runtime': 3329.0328, 'train_samples_per_second': 130.398, 'train_steps_per_secon
d': 8.153, 'total_flos': 2.87154521229312e+16, 'train_loss': 0.10907475394893412,
'epoch': 10.0})
```

In [16]:
```
multilabel_pipeline = pipeline(
    "text-classification",
    model=model,
    tokenizer=tokenizer,
    top_k=None,  # return all pedictions first
)
```

```
Device set to use cuda:0
```

In [17]:
```
#result = multilabel_pipeline("I simply can't stand when my computer freezes")
result = multilabel_pipeline("I hate you, but i also like you")
#result = multilabel_pipeline("I'm Not Mad")
result
```

Out[17]:
```
[[{'label': 'LABEL_18', 'score': 0.38844403624534607},
  {'label': 'LABEL_25', 'score': 0.11596072465181351},
  {'label': 'LABEL_3', 'score': 0.07450026273727417},
  {'label': 'LABEL_27', 'score': 0.0696682259440422},
  {'label': 'LABEL_2', 'score': 0.06548520922660828},
  {'label': 'LABEL_1', 'score': 0.058884698897600174},
  {'label': 'LABEL_9', 'score': 0.03796369582414627},
  {'label': 'LABEL_4', 'score': 0.03468417376279831},
  {'label': 'LABEL_17', 'score': 0.03378022834658623},
  {'label': 'LABEL_11', 'score': 0.027637720108032227},
  {'label': 'LABEL_8', 'score': 0.02742854319512844},
  {'label': 'LABEL_0', 'score': 0.02409944497048855},
  {'label': 'LABEL_14', 'score': 0.02025469020009041},
  {'label': 'LABEL_10', 'score': 0.018369637429714203},
  {'label': 'LABEL_5', 'score': 0.016657039523124695},
  {'label': 'LABEL_24', 'score': 0.012576347216963768},
  {'label': 'LABEL_22', 'score': 0.012431650422513485},
  {'label': 'LABEL_12', 'score': 0.01164222415536642},
  {'label': 'LABEL_20', 'score': 0.007204769179224968},
  {'label': 'LABEL_16', 'score': 0.005842605140060186},
  {'label': 'LABEL_19', 'score': 0.005598248913884163},
  {'label': 'LABEL_13', 'score': 0.005139610730111599},
  {'label': 'LABEL_21', 'score': 0.002540992572903633},
  {'label': 'LABEL_23', 'score': 0.002540992572903633},
  {'label': 'LABEL_7', 'score': 0.0023141263518482447},
  {'label': 'LABEL_15', 'score': 0.0019800399895757437},
  {'label': 'LABEL_26', 'score': 0.0017007224960252643},
  {'label': 'LABEL_6', 'score': 0.0015367271844297647}]]
```

In [18]:
```
def get_emotion(pred):
    """
    Gets the int number after the _ in "LABEL_X"
    e.g.: LABEL_18 yields just the integer 18
    We use this integer as index to get the original emotion name
    """
    pred_emotion_idx = int(pred["label"].split('_')[1])
    return emotions[pred_emotion_idx]

predicted_emotions = [get_emotion(pred) for pred in result[0] if pred["score"] > 0.
predicted_emotions
```

Out[18]:
```
['love', 'sadness']
```

In [19]:
```
# === Per-label threshold search (validation) and test evaluation ===
import numpy as np
import pandas as pd
from sklearn.metrics import f1_score, precision_recall_fscore_support, accuracy_sco

def _sigmoid(x): return 1 / (1 + np.exp(-x))

def _predict_split(trainer, split_ds):
    out = trainer.predict(split_ds)
    y_true = out.label_ids
    y_score = _sigmoid(out.predictions)
    if y_score.ndim == 1: y_score = y_score.reshape(-1, 1)
```

```
    return y_true, y_score

# 1) Get validation scores
y_true_val, y_score_val = _predict_split(trainer, tokenized_dataset["validation"])
L = y_true_val.shape[1]
if "label_names" not in globals():
    label_names = [f"L{i}" for i in range(L)]

# 2) Grid-search a separate threshold for each label
grid = np.round(np.arange(0.05, 0.95 + 1e-9, 0.01), 2)
best_thr_per_label = np.zeros(L)
best_f1_per_label  = np.zeros(L)

for j in range(L):
    yj_true  = y_true_val[:, j]
    yj_score = y_score_val[:, j]
    best_f1, best_t = -1, 0.5
    for t in grid:
        yj_pred = (yj_score >= t).astype(int)
        f1 = f1_score(yj_true, yj_pred, zero_division=0)
        if f1 > best_f1:
            best_f1, best_t = f1, t
    best_thr_per_label[j] = best_t
    best_f1_per_label[j]  = best_f1

thr_table = pd.DataFrame({
    "label": label_names,
    "best_thr": best_thr_per_label,
    "val_f1_at_best_thr": best_f1_per_label
}).sort_values("val_f1_at_best_thr", ascending=False).reset_index(drop=True)
display(thr_table.head(15))

# 3) Evaluate on TEST using per-label thresholds
y_true_test, y_score_test = _predict_split(trainer, tokenized_dataset["test"])
y_pred_test = (y_score_test >= best_thr_per_label[None, :]).astype(int)

# Summary metrics
subset_acc = accuracy_score(y_true_test, y_pred_test)
micro_f1   = f1_score(y_true_test, y_pred_test, average="micro", zero_division=0)
macro_f1   = f1_score(y_true_test, y_pred_test, average="macro", zero_division=0)
samples_f1 = f1_score(y_true_test, y_pred_test, average="samples", zero_division=0)

print("=== TEST (per-label thresholds) ===")
print(f"subset_accuracy: {subset_acc:.4f}")
print(f"micro_f1:       {micro_f1:.4f}")
print(f"macro_f1:       {macro_f1:.4f}")
print(f"samples_f1:     {samples_f1:.4f}")

# Per-label breakdown
p, r, f, s = precision_recall_fscore_support(y_true_test, y_pred_test, average=None
per_label = pd.DataFrame({
    "label": label_names, "precision": p, "recall": r, "f1": f, "support": s, "thr_
}).sort_values("support", ascending=False).reset_index(drop=True)
display(per_label.head(20))
```

	label	best_thr	val_f1_at_best_thr
0	L15	0.61	0.918605
1	L18	0.21	0.770318
2	L1	0.36	0.767742
3	L24	0.26	0.753425
4	L0	0.40	0.711340
5	L27	0.27	0.670946
6	L20	0.35	0.576177
7	L25	0.27	0.531148
8	L17	0.31	0.529968
9	L7	0.36	0.517647
10	L14	0.15	0.467532
11	L2	0.28	0.465558
12	L26	0.29	0.435556
13	L8	0.16	0.428571
14	L10	0.14	0.386127

```
=== TEST (per-label thresholds) ===
subset_accuracy: 0.3414
micro_f1:       0.5415
macro_f1:       0.3990
samples_f1:     0.5513
```

	label	precision	recall	f1	support	thr_used
0	L27	0.586831	0.797985	0.676310	1787	0.27
1	L0	0.653924	0.644841	0.649351	504	0.40
2	L15	0.945289	0.883523	0.913363	352	0.61
3	L4	0.352941	0.307692	0.328767	351	0.20
4	L3	0.243197	0.446875	0.314978	320	0.16
5	L7	0.454768	0.654930	0.536797	284	0.36
6	L10	0.269297	0.588015	0.369412	267	0.14
7	L1	0.788889	0.806818	0.797753	264	0.36
8	L18	0.727915	0.865546	0.790787	238	0.21
9	L2	0.423256	0.459596	0.440678	198	0.28
10	L20	0.666667	0.473118	0.553459	186	0.35
11	L17	0.612676	0.540373	0.574257	161	0.31
12	L25	0.444444	0.461538	0.452830	156	0.27
13	L6	0.296748	0.477124	0.365915	153	0.22
14	L9	0.156923	0.337748	0.214286	151	0.11
15	L22	0.145985	0.137931	0.141844	145	0.07
16	L26	0.472527	0.304965	0.370690	141	0.29
17	L5	0.263636	0.429630	0.326761	135	0.14
18	L11	0.284615	0.300813	0.292490	123	0.13
19	L13	0.311828	0.281553	0.295918	103	0.14

In [20]:
```
# === Surface interesting errors on TEST ===
import pandas as pd
import numpy as np

# Ensure we have y_true_test/y_score_test/y_pred_test from A1 cell; otherwise recom
if 'y_true_test' not in globals() or 'y_pred_test' not in globals():
    y_true_test, y_score_test = _predict_split(trainer, tokenized_dataset["test"])
    y_pred_test = (y_score_test >= 0.5).astype(int)

texts_test = tokenized_dataset["test"]["text"]

def labels_to_str(row):
    return ", ".join(np.array(label_names)[np.where(row==1)[0]])

df_err = pd.DataFrame({
    "text": texts_test,
    "true": [labels_to_str(y_true_test[i]) for i in range(len(texts_test))],
```

```
    "pred": [labels_to_str(y_pred_test[i]) for i in range(len(texts_test))]
})
df_err = df_err[df_err["true"] != df_err["pred"]]

print("Total misclassified samples:", len(df_err))
display(df_err.sample(8, random_state=42))
```

```
Total misclassified samples: 3574
```

	text	true	pred
2719	Most vancouverites know of these scams. Especi...	L27	L10, L27
4129	Yep California, I would've been surprised if t...	L4	L26
48	You take that back right now! [NAME] does not ...	L27	L3, L10, L27
733	Actually maybe the OP is not an INTP, have you...	L7	L6, L27
4781	What type would you classify House from House ...	L27	L7, L27
5332	Ah!! Was not expecting to read that this morni...	L0, L1, L22	L1
3762	To be fair, I will also continue to jerk off t...	L0, L4	L0
708	Resetting a dislocated knee hurts like hell bu...	L23	L9, L25

In [21]:
```
# === Best & worst labels by F1 (TEST) ===
from sklearn.metrics import precision_recall_fscore_support

p, r, f, s = precision_recall_fscore_support(y_true_test, y_pred_test, average=None
table = pd.DataFrame({"label": label_names, "precision": p, "recall": r, "f1": f, "
best5  = table.sort_values("f1", ascending=False).head(5)
worst5 = table.sort_values("f1", ascending=True).head(5)
print("Top-5 labels by F1:")
display(best5)
print("Bottom-5 labels by F1:")
display(worst5)
```

```
Top-5 labels by F1:
```

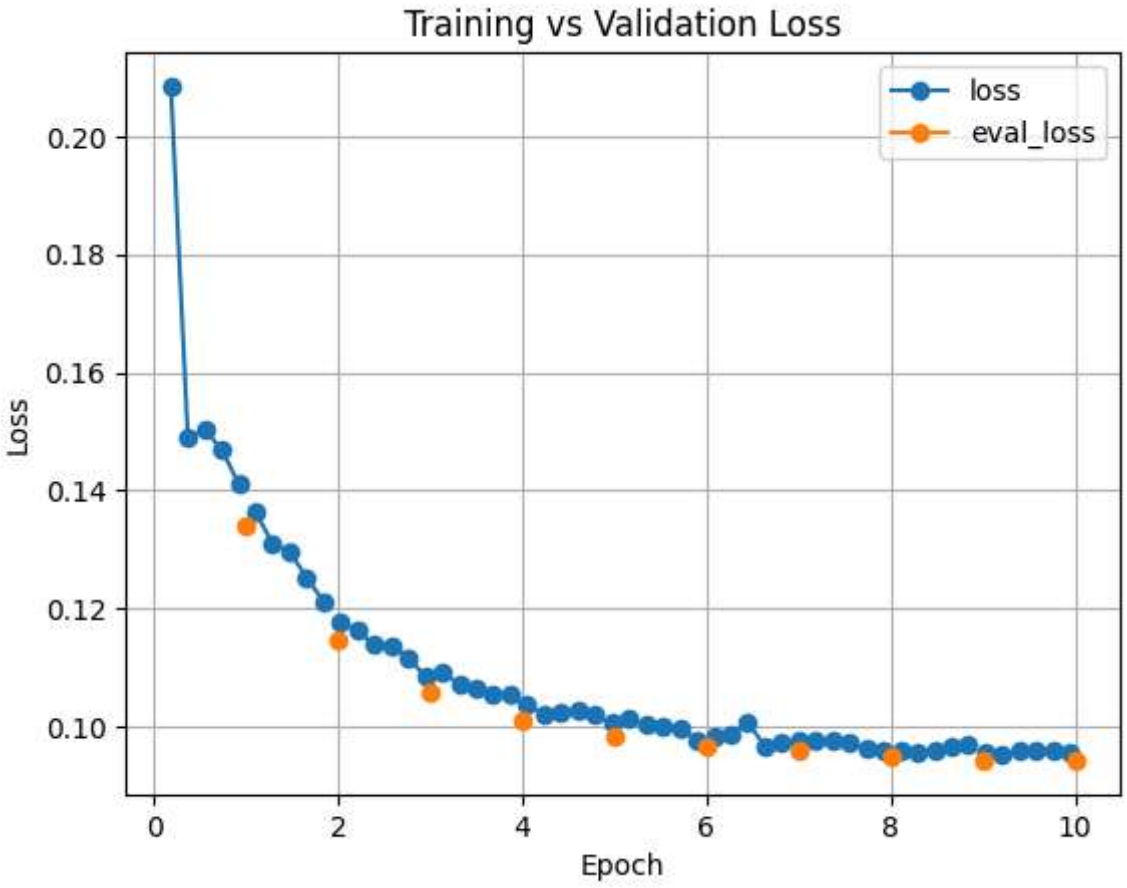
	label	precision	recall	f1	support
15	L15	0.945289	0.883523	0.913363	352
1	L1	0.788889	0.806818	0.797753	264
18	L18	0.727915	0.865546	0.790787	238
27	L27	0.586831	0.797985	0.676310	1787
0	L0	0.653924	0.644841	0.649351	504

```
Bottom-5 labels by F1:
```

	label	precision	recall	f1	support
21	L21	0.000000	0.000000	0.000000	16
23	L23	0.000000	0.000000	0.000000	11
16	L16	0.000000	0.000000	0.000000	6
19	L19	0.120000	0.130435	0.125000	23
22	L22	0.145985	0.137931	0.141844	145

In [22]:
```python
# === Plot learning curves from Trainer logs ===
import pandas as pd
import matplotlib.pyplot as plt

logs = pd.DataFrame(trainer.state.log_history)
# Keep useful keys only
keys = ["epoch", "loss", "eval_loss", "eval_f1", "eval_accuracy"]
curves = logs[[k for k in keys if k in logs.columns]].dropna(how="all")

# Loss curve
plt.figure()
curves.plot(x="epoch", y=[c for c in ["loss","eval_loss"] if c in curves.columns],
plt.title("Training vs Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True)
plt.show()
```

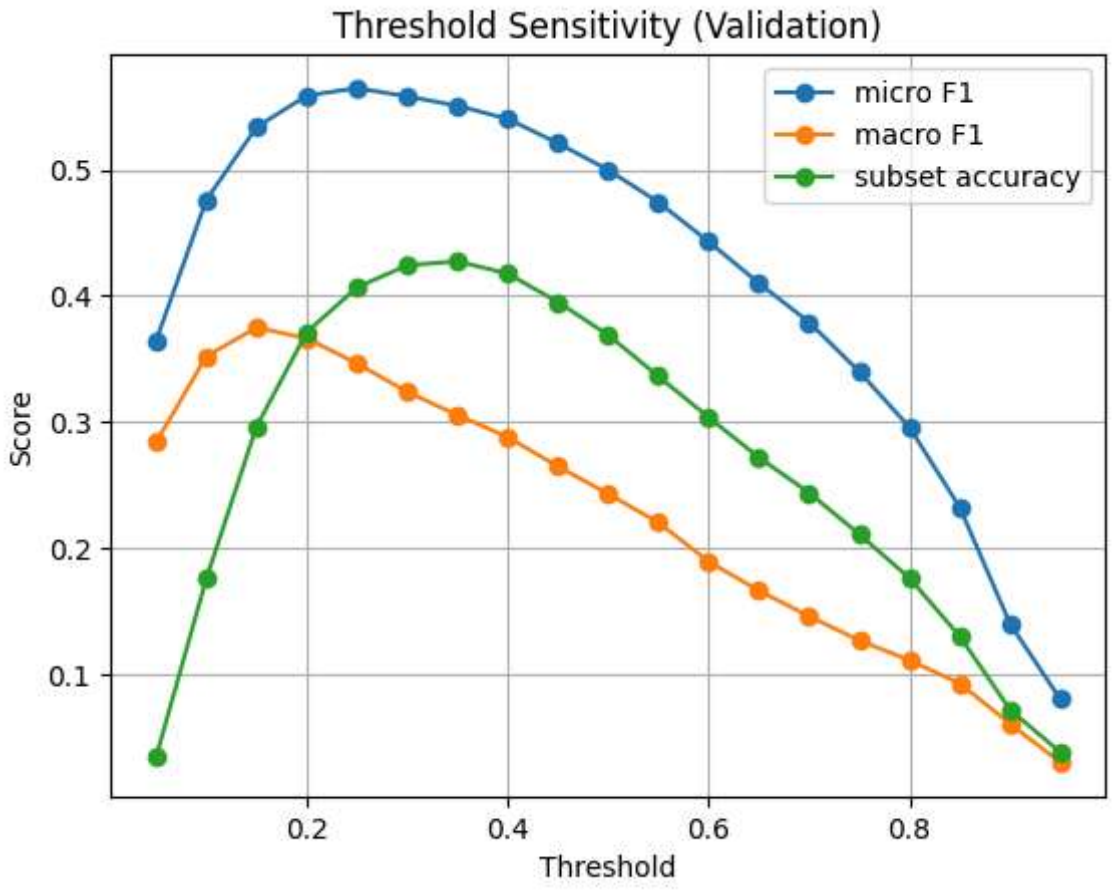
```
<Figure size 640x480 with 0 Axes>
```



In [23]:
```python
# === Efficiency snapshot ===
import torch, numpy as np, pandas as pd

def count_params(m):
    total = sum(p.numel() for p in m.parameters())
    trainable = sum(p.numel() for p in m.parameters() if p.requires_grad)
    return total, trainable

total_p, trainable_p = count_params(model)
runtime_sec = float(getattr(trainer.state, "train_runtime", np.nan))
steps_per_sec = float(getattr(trainer.state, "train_steps_per_second", np.nan))
samples_per_sec = float(getattr(trainer.state, "train_samples_per_second", np.nan))

try:
    vram_mb = torch.cuda.max_memory_allocated() / (1024**2)
except:
    vram_mb = np.nan

eff = pd.DataFrame([{
    "strategy": "current_model",
    "total_params": total_p,
    "trainable_params": trainable_p,
    "train_runtime_s": runtime_sec,
    "steps_per_s": steps_per_sec,
    "samples_per_s": samples_per_sec,
    "approx_max_VRAM_MB": vram_mb
}])
display(eff)
```

	strategy	total_params	trainable_params	train_runtime_s	steps_per_s	samples_per_s
0	current_model	109967672	463900	NaN	NaN	NaN

In [ ]:
```python
# === Single-threshold sensitivity (VAL) ===
import numpy as np, matplotlib.pyplot as plt
from sklearn.metrics import f1_score, accuracy_score

y_true_val, y_score_val = _predict_split(trainer, tokenized_dataset["validation"])
ths = np.linspace(0.05, 0.95, 19)
micro_f1s, macros, accs = [], [], []

for t in ths:
    y_pred = (y_score_val >= t).astype(int)
    micro_f1s.append(f1_score(y_true_val, y_pred, average="micro", zero_division=0))
    macros.append(f1_score(y_true_val, y_pred, average="macro", zero_division=0))
    accs.append(accuracy_score(y_true_val, y_pred))

plt.figure()
plt.plot(ths, micro_f1s, marker="o", label="micro F1")
plt.plot(ths, macros, marker="o", label="macro F1")
plt.plot(ths, accs, marker="o", label="subset accuracy")
plt.xlabel("Threshold")
plt.ylabel("Score")
plt.title("Threshold Sensitivity (Validation)")
plt.grid(True)
plt.legend()
plt.show()
```




```
The Kernel crashed while executing code in the current cell or a previous cell.

Please review the code in the cell(s) to identify a possible cause of the failure.

Click <a href='https://aka.ms/vscodeJupyterKernelCrash'>here</a> for more info.

View Jupyter <a href='command:jupyter.viewOutput'>log</a> for further details.
```