```
# 0) Install libraries
!pip install -q sentence-transformers datasets transformers scikit-learn
```

```
# 1) Imports
import os, zipfile, math
import pandas as pd
import numpy as np
from tqdm.auto import tqdm
from sklearn import model_selection
from sklearn.metrics import f1_score, precision_recall_fscore_support
import torch

from sentence_transformers import SentenceTransformer, InputExample, util, losses
from sentence_transformers.cross_encoder import CrossEncoder
from torch.utils.data import DataLoader
```

```
# 2) Settings
RANDOM_STATE = 42
SAMPLE = False
SAMPLE_N = 20000
BENCHMARK_MODEL = "all-MiniLM-L6-v2"
CROSS_ENCODER_MODEL = "cross-encoder/ms-marco-MiniLM-L-6-v2"
EPOCHS = 1
BATCH_SIZE = 32
CROSS_BATCH = 16
MAX_TRAIN_SAMPLES = None

device = "cuda" if torch.cuda.is_available() else "cpu"
print("Device:", device)
```

```
Device: cuda
```

```
# 3) Load dataset (train.csv.zip and test.csv.zip uploaded by you)
def load_from_zip(zip_path, filename):
    with zipfile.ZipFile(zip_path, 'r') as z:
        z.extractall("./data_extracted")
    return pd.read_csv(f"./data_extracted/{filename}")

print("Loading dataset... (unzipping and reading)")
train_df = load_from_zip("train.csv.zip", "train.csv")
test_public_df = load_from_zip("test.csv.zip", "test.csv")
print("Train shape:", train_df.shape, "| Test shape:", test_public_df.shape)
```

```
Loading dataset... (unzipping and reading)
Train shape: (404290, 6) | Test shape: (2345796, 3)
```

```
# 4) Clean and keep relevant columns
train_df = train_df.dropna(subset=["question1", "question2"]).reset_index(drop=True)
train_df = train_df.rename(columns={"is_duplicate":"label"})
train_df = train_df[["question1","question2","label"]].copy()
train_df["label"] = train_df["label"].astype(int)

# Optional sampling
if SAMPLE:
    train_df = train_df.sample(n=min(SAMPLE_N, len(train_df)), random_state=RANDOM_STATE).reset_index(drop=True)
print("Using dataset size:", len(train_df))
```

```
Using dataset size: 404287
```

```
# 5) Train/Val/Test split
test_size = 0.10
val_size = 0.05
train_val_df, test_df = model_selection.train_test_split(
    train_df, test_size=test_size, stratify=train_df["label"], random_state=RANDOM_STATE
)
val_relative = val_size / (1 - test_size)
train_df, val_df = model_selection.train_test_split(
    train_val_df, test_size=val_relative, stratify=train_val_df["label"], random_state=RANDOM_STATE
)

print(f"Train: {len(train_df)} | Val: {len(val_df)} | Test: {len(test_df)}")
```

```
Train: 343643 | Val: 20215 | Test: 40429
```

```
# ================================================================
# Helper functions
# ================================================================
def maybe_subsample(df, max_samples):
    if max_samples is None:
        return df
    return df.sample(n=min(max_samples, len(df)), random_state=RANDOM_STATE).reset_index(drop=True)

def tune_threshold_and_eval_on_test(model, val_df, test_df, batch_size=64):
    q1_val, q2_val = val_df["question1"].tolist(), val_df["question2"].tolist()
    y_val = val_df["label"].tolist()
    q1_test, q2_test = test_df["question1"].tolist(), test_df["question2"].tolist()
    y_test = test_df["label"].tolist()

    emb_q1_val = model.encode(q1_val, convert_to_tensor=True, batch_size=batch_size)
    emb_q2_val = model.encode(q2_val, convert_to_tensor=True, batch_size=batch_size)
    emb_q1_test = model.encode(q1_test, convert_to_tensor=True, batch_size=batch_size)
    emb_q2_test = model.encode(q2_test, convert_to_tensor=True, batch_size=batch_size)

    cos_val = util.cos_sim(emb_q1_val, emb_q2_val).diagonal().cpu().numpy()
    cos_test = util.cos_sim(emb_q1_test, emb_q2_test).diagonal().cpu().numpy()

    # Tune threshold
    best_thr, best_f1 = 0.5, -1
    for thr in np.linspace(0, 1, 101):
        preds = (cos_val >= thr).astype(int)
        f1 = f1_score(y_val, preds)
        if f1 > best_f1:
            best_f1, best_thr = f1, thr

    test_preds = (cos_test >= best_thr).astype(int)
    precision, recall, f1, _ = precision_recall_fscore_support(y_test, test_preds, average='binary', zero_division=0)
    return {"best_val_f1": best_f1, "best_threshold": best_thr,
            "test_precision": precision, "test_recall": recall, "test_f1": f1}

def eval_cross_encoder(ce_model, val_df, test_df):
    val_pairs = list(zip(val_df["question1"], val_df["question2"]))
    test_pairs = list(zip(test_df["question1"], test_df["question2"]))
    y_val, y_test = val_df["label"].tolist(), test_df["label"].tolist()

    val_scores = ce_model.predict(val_pairs)
    test_scores = ce_model.predict(test_pairs)

    best_thr, best_f1 = 0.5, -1
    for thr in np.linspace(min(val_scores), max(val_scores), 101):
        preds = (np.array(val_scores) >= thr).astype(int)
        f1 = f1_score(y_val, preds)
        if f1 > best_f1:
            best_f1, best_thr = f1, thr

    test_preds = (np.array(test_scores) >= best_thr).astype(int)
    precision, recall, f1, _ = precision_recall_fscore_support(y_test, test_preds, average='binary', zero_division=0)
    return {"best_val_f1": best_f1, "best_threshold": best_thr,
            "test_precision": precision, "test_recall": recall, "test_f1": f1}


# Training helper for bi-encoders
def train_biencoder(train_df, model_name, loss_name="CosineSimilarityLoss", epochs=1, batch_size=32, max_train_samples=None):
    train_use = maybe_subsample(train_df, max_train_samples)
    train_examples = []
    for _, r in train_use.iterrows():
        if loss_name in ["ContrastiveLoss", "CosineSimilarityLoss"]:
            train_examples.append(InputExample(texts=[str(r["question1"]), str(r["question2"])], label=float(r["label"])))
        else:
            train_examples.append(InputExample(texts=[str(r["question1"]), str(r["question2"])]))

    model = SentenceTransformer(model_name, device=device)
    train_dataloader = DataLoader(train_examples, shuffle=True, batch_size=batch_size)

    if loss_name == "CosineSimilarityLoss":
        train_loss = losses.CosineSimilarityLoss(model)
    elif loss_name == "ContrastiveLoss":
        train_loss = losses.ContrastiveLoss(model)
    else:
        train_loss = losses.MultipleNegativesRankingLoss(model)

    model.fit(train_objectives=[(train_dataloader, train_loss)], epochs=epochs, show_progress_bar=True)
    return model


# ============================================================
# Experiments
# ============================================================
from sentence_transformers import SentenceTransformer, InputExample, losses, CrossEncoder
from torch.utils.data import DataLoader
```

```
results = []

# -------------------------
# Experiment 1: Pretrained Bi-encoder
# -------------------------
print("\n=== Experiment 1: Pretrained Bi-encoder ===")
bench_model = SentenceTransformer(BENCHMARK_MODEL, device=device)
bench_res = tune_threshold_and_eval_on_test(bench_model, val_df, test_df)
bench_res.update({"experiment": "benchmark", "model": BENCHMARK_MODEL})
results.append(bench_res)
```

```

=== Experiment 1: Pretrained Bi-encoder ===
```

```
# -------------------------
# Experiment 2: Bi-encoder CosineLoss
# -------------------------
print("\n=== Experiment 2: Bi-encoder CosineLoss ===")

# prepare training data
train_samples = [
    InputExample(texts=[q1, q2], label=float(label))
    for q1, q2, label in zip(train_df["question1"], train_df["question2"], train_df["label"])
]

train_dataloader = DataLoader(train_samples, shuffle=True, batch_size=BATCH_SIZE)

cos_model = SentenceTransformer(BENCHMARK_MODEL, device=device)
train_loss = losses.CosineSimilarityLoss(cos_model)

cos_model.fit(
    train_objectives=[(train_dataloader, train_loss)],
    epochs=EPOCHS,
    warmup_steps=1000,
    show_progress_bar=True,
)

cos_res = tune_threshold_and_eval_on_test(cos_model, val_df, test_df)
cos_res.update({"experiment": "bi_cosine", "model": BENCHMARK_MODEL})
results.append(cos_res)
```

```
=== Experiment 2: Bi-encoder CosineLoss ===

[10739/10739 13:49, Epoch 1/1]
 Step  Training Loss
  500       0.190000
 1000       0.146800
 1500       0.138000
 2000       0.129500
 2500       0.125900
 3000       0.122900
 3500       0.122300
 4000       0.122700
 4500       0.120600
 5000       0.120000
 5500       0.115600
 6000       0.117800
 6500       0.116600
 7000       0.115200
 7500       0.113500
 8000       0.114500
 8500       0.114400
 9000       0.115400
 9500       0.112600
10000       0.111800
10500       0.112200
```

```
# -------------------------
# Experiment 3: Cross-encoder
# -------------------------
print("\n=== Experiment 3: Cross-encoder ===")

from sentence_transformers import InputExample
from torch.utils.data import DataLoader

# prepare training samples
train_samples = [
    InputExample(texts=[q1, q2], label=float(label))
    for q1, q2, label in zip(train_df["question1"], train_df["question2"], train_df["label"])
]

# dataloader with batch size
train_dataloader = DataLoader(train_samples, shuffle=True, batch_size=CROSS_BATCH)

# init cross-encoder
ce_model = CrossEncoder(CROSS_ENCODER_MODEL, num_labels=1, device=device)

# fit model
ce_model.fit(
    train_dataloader=train_dataloader,
    epochs=EPOCHS,
    warmup_steps=1000,
    show_progress_bar=True,
)

# evaluate
ce_res = eval_cross_encoder(ce_model, val_df, test_df)
ce_res.update({"experiment": "cross_encoder", "model": CROSS_ENCODER_MODEL})
results.append(ce_res)
```

```
=== Experiment 3: Cross-encoder ===
```

[21478/21478 13:21, Epoch 1/1]

Step	Training Loss
500	0.632200
1000	0.425000
1500	0.395400
2000	0.378900
2500	0.367700
3000	0.359900
3500	0.348700
4000	0.334000
4500	0.328800
5000	0.334500
5500	0.321500
6000	0.318100
6500	0.325500
7000	0.316200
7500	0.308000
8000	0.325800
8500	0.313200
9000	0.307000
9500	0.320200
10000	0.304600
10500	0.303100
11000	0.295900
11500	0.311900
12000	0.309000
12500	0.291100
13000	0.302000
13500	0.301600
14000	0.296800
14500	0.293100
15000	0.277600
15500	0.299600
16000	0.286400
16500	0.283700
17000	0.290600
17500	0.305900
18000	0.289600
18500	0.282500
19000	0.284300
19500	0.284200

```
# =================================================================
# Final Results
# =================================================================
print("\n=== All Experiment Results ===")
for r in results:
    print(r)
```

```
=== All Experiment Results ===
{'best_val_f1': 0.7403302420266908, 'best_threshold': np.float64(0.75), 'test_precision': 0.6351371168793079, 'test_reca
{'best_val_f1': 0.8174355711104041, 'best_threshold': np.float64(0.6), 'test_precision': 0.7780802032299038, 'test_recal
{'best_val_f1': 0.8462950072770993, 'best_threshold': np.float32(-0.5954857), 'test_precision': 0.802461759082218, 'test
```