Import libraries and dependencies for data, models, and evaluation

```
import pandas as pd
import numpy as np
import os
from sklearn.model_selection import train_test_split
from sentence_transformers import SentenceTransformer, util,
InputExample, losses, CrossEncoder
from sklearn.metrics import f1_score
from torch.utils.data import DataLoader
os.environ["WANDB_DISABLED"] = "true"
```

Loading the dataset, creating a 50k stratified subset, and splitting it

```
df = pd.read_csv("train.csv")

subset_size = 50000
df_subset, _ = train_test_split(
    df,
    train_size=subset_size,
    stratify=df["is_duplicate"],
    random_state=42
)

train, temp = train_test_split(
    df_subset,
    test_size=0.2,
    stratify=df_subset["is_duplicate"],
    random_state=42
)
valid, test = train_test_split(
    temp,
    test_size=0.5,
    stratify=temp["is_duplicate"],
    random_state=42
)

os.makedirs("splits", exist_ok=True)

train.to_csv("splits/train.csv", index=False)
valid.to_csv("splits/valid.csv", index=False)
test.to_csv("splits/test.csv", index=False)

print("Final sizes:", len(train), len(valid), len(test))
```

Baseline Model: Encode test questions with a pre-trained bi-encoder

```
test = pd.read_csv("splits/test.csv")

model = SentenceTransformer("sentence-transformers/all-MiniLM-L6-v2")

emb1 = model.encode(test["question1"].tolist(), batch_size=128,
convert_to_numpy=True)
emb2 = model.encode(test["question2"].tolist(), batch_size=128,
convert_to_numpy=True)

sims = util.cos_sim(emb1, emb2).diagonal()

best_f1, best_thr = 0, 0
for thr in [i/100 for i in range(-100, 101)]:
    sims = util.cos_sim(emb1, emb2).diagonal().cpu().numpy()
    preds = (sims >= thr).astype(int)
    f1 = f1_score(test["is_duplicate"], preds)
    if f1 > best_f1:
        best_f1, best_thr = f1, thr

print(f"[Baseline] Test F1={best_f1:.4f} at threshold={best_thr:.2f}")
```

Training and Evaluating Bi-Encoders

Fine-tuning three different bi-encoders with specific loss functions and base models:

- **Cosine Similarity Loss** using `all-MiniLM-L6-v2`
- **Contrastive Loss** using `paraphrase-MiniLM-L6-v2`
- **Multiple Negatives Ranking Loss (MNR)** using `all-mpnet-base-v2`

```
train = pd.read_csv("splits/train.csv")
valid = pd.read_csv("splits/valid.csv")
test = pd.read_csv("splits/test.csv")

def to_examples(df):
    return [InputExample(texts=[row["question1"], row["question2"]],
label=float(row["is_duplicate"])) for _, row in df.iterrows()]

train_examples = to_examples(train)
valid_examples = to_examples(valid)
test_examples = to_examples(test)

def run_biencoder(loss_type, base_model, epochs=2, batch_size=32,
lr=2e-5):
    model = SentenceTransformer(base_model)

    if loss_type == "mnr":
```

```
        train_loss = losses.MultipleNegativesRankingLoss(model)
    elif loss_type == "cos":
        for ex in train_examples:
            ex.label = 1.0 if ex.label == 1.0 else -1.0
        train_loss = losses.CosineSimilarityLoss(model)
    elif loss_type == "contrastive":
        train_loss = losses.OnlineContrastiveLoss(
            model,

distance_metric=losses.SiameseDistanceMetric.COSINE_DISTANCE,
            margin=0.5
        )
    else:
        raise ValueError("loss_type must be one of: mnr, cos,
contrastive")

    train_dataloader = DataLoader(train_examples, shuffle=True,
batch_size=batch_size)

    model.fit(
        train_objectives=[(train_dataloader, train_loss)],
        epochs=epochs,
        warmup_steps=100,
        optimizer_params={'lr': lr},
        show_progress_bar=True
    )

    # Evaluate
    def evaluate(model, examples, lt):
        q1 = [ex.texts[0] for ex in examples]
        q2 = [ex.texts[1] for ex in examples]
        labels = [int(ex.label) if lt != "cos" else (1 if ex.label ==
1.0 else 0) for ex in examples]

        emb1 = model.encode(q1, batch_size=128, convert_to_numpy=True)
        emb2 = model.encode(q2, batch_size=128, convert_to_numpy=True)
        sims = util.cos_sim(emb1, emb2).diagonal().cpu().numpy()

        best_f1, best_thr = 0, 0
        for thr in [i/100 for i in range(-100, 101)]:
            preds = (sims >= thr).astype(int)
            f1 = f1_score(labels, preds)
            if f1 > best_f1:
                best_f1, best_thr = f1, thr
        return best_f1, best_thr

    val_f1, thr = evaluate(model, valid_examples, loss_type)
    test_f1, _ = evaluate(model, test_examples, loss_type)
    print(f"[{loss_type}] Validation F1={val_f1:.4f} | Test
F1={test_f1:.4f} at threshold={thr:.2f}")
```

```
    return model, test_f1

cos_model, cos_f1 = run_biencoder("cos", "sentence-transformers/all-
MiniLM-L6-v2")
contrast_model, contrast_f1 = run_biencoder("contrastive", "sentence-
transformers/paraphrase-MiniLM-L6-v2")
mnr_model, mnr_f1 = run_biencoder("mnr", "sentence-transformers/all-
mpnet-base-v2")
```

Cross-Encoder Training & Evaluation

Fine-tuning `ms-marco-MiniLM-L-6-v2` on the training split

```
train_samples = [
    InputExample(texts=[row["question1"], row["question2"]],
label=float(row["is_duplicate"]))
    for _, row in train.iterrows()
]
valid_samples = [
    (row["question1"], row["question2"], int(row["is_duplicate"]))
    for _, row in valid.iterrows()
]
test_samples = [
    (row["question1"], row["question2"], int(row["is_duplicate"]))
    for _, row in test.iterrows()
]

train_dataloader = DataLoader(train_samples, shuffle=True,
batch_size=16)


ce_model = CrossEncoder("cross-encoder/ms-marco-MiniLM-L-6-v2",
num_labels=1)

ce_model.fit(
    train_dataloader=train_dataloader,
    epochs=1,
    warmup_steps=100,
    show_progress_bar=True
)


def evaluate_ce(model, samples):
    texts = [(q1, q2) for q1, q2, _ in samples]
    labels = [lbl for _, _, lbl in samples]

    scores = model.predict(texts)

    best_f1, best_thr = 0, 0
```

```
    for thr in np.linspace(0, 1, 101):
        preds = (scores >= thr).astype(int)
        f1 = f1_score(labels, preds)
        if f1 > best_f1:
            best_f1, best_thr = f1, thr
    return best_f1, best_thr

val_f1, thr = evaluate_ce(ce_model, valid_samples)
test_f1, _ = evaluate_ce(ce_model, test_samples)

print(f"[CrossEncoder] Validation F1={val_f1:.4f} | Test
F1={test_f1:.4f} at threshold={thr:.2f}")
```

F1 Score Eval

```
results = {
    "Baseline": "-",
    "Bi-encoder (Cosine)": cos_f1,
    "Bi-encoder (Contrastive)": contrast_f1,
    "Bi-encoder (MNR)": mnr_f1,
    "Cross-encoder": test_f1
}
print(results)
```