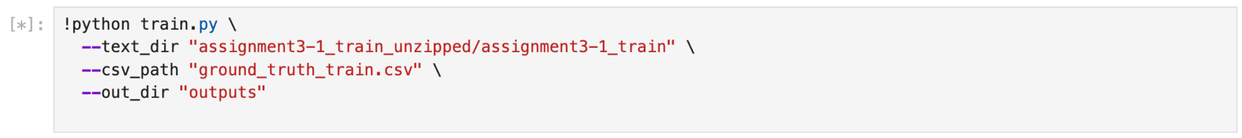
TRAINING CODES: APPENDIX

This appendix presents the training implementation used to preprocess the dataset, perform an 80/20 stratified train, validation split, and train the RNN/LSTM-based neural network to jointly predict news topic and political ideology labels.

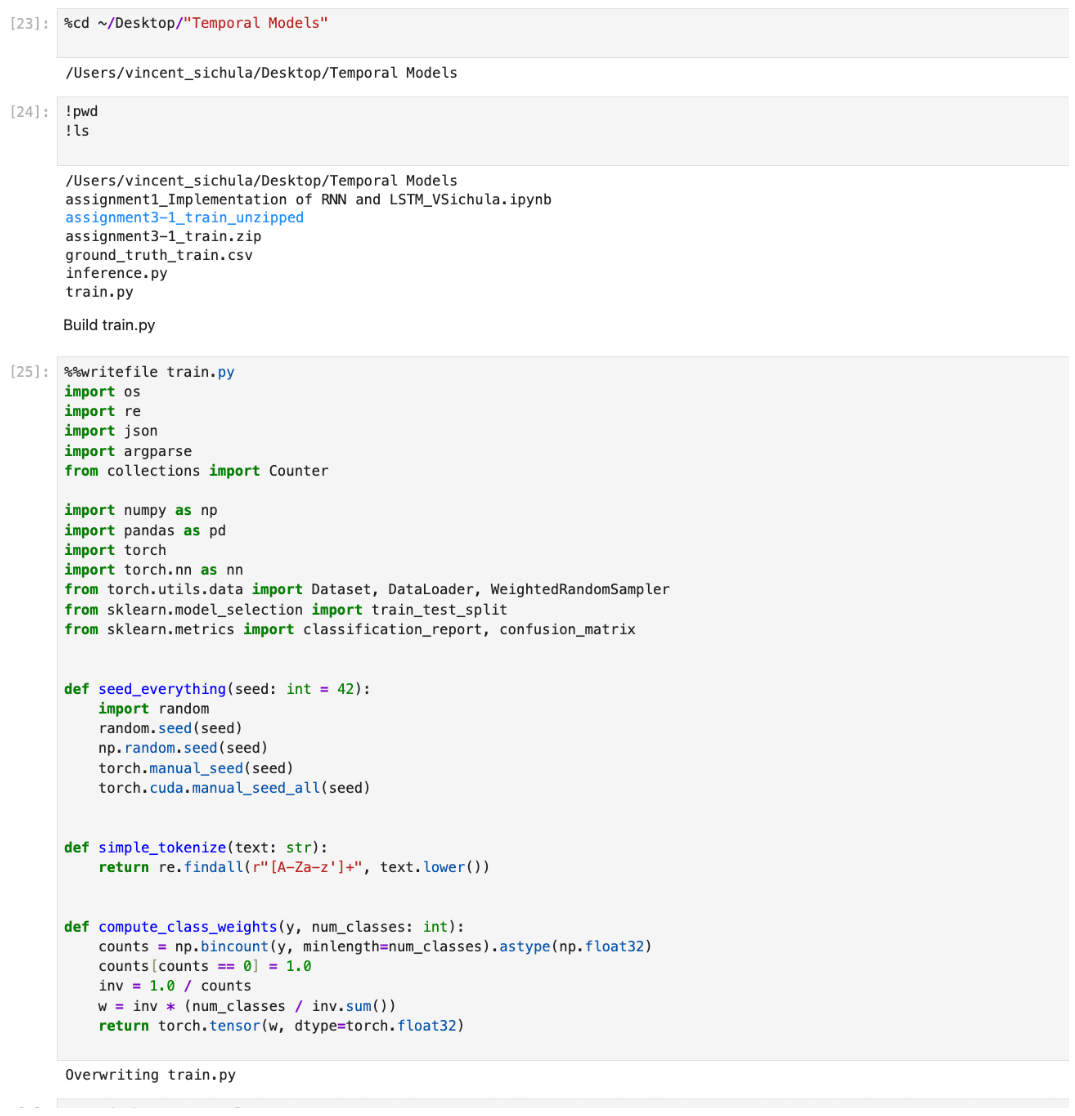
0. Training command (Jupyter cell)

The training process was executed from a Jupyter notebook using a shell command. The command below specifies the path to the folder containing the news articles, the ground truth CSV file, and the output directory for model checkpoints and reports.



[*]:
```
!python train.py \
  --text_dir "assignment3-1_train_unzipped/assignment3-1_train" \
  --csv_path "ground_truth_train.csv" \
  --out_dir "outputs"
```

1. Imports and utilities



[23]:
```
%cd ~/Desktop/"Temporal Models"
```

```
/Users/vincent_sichula/Desktop/Temporal Models
```

[24]:
```
!pwd
!ls
```

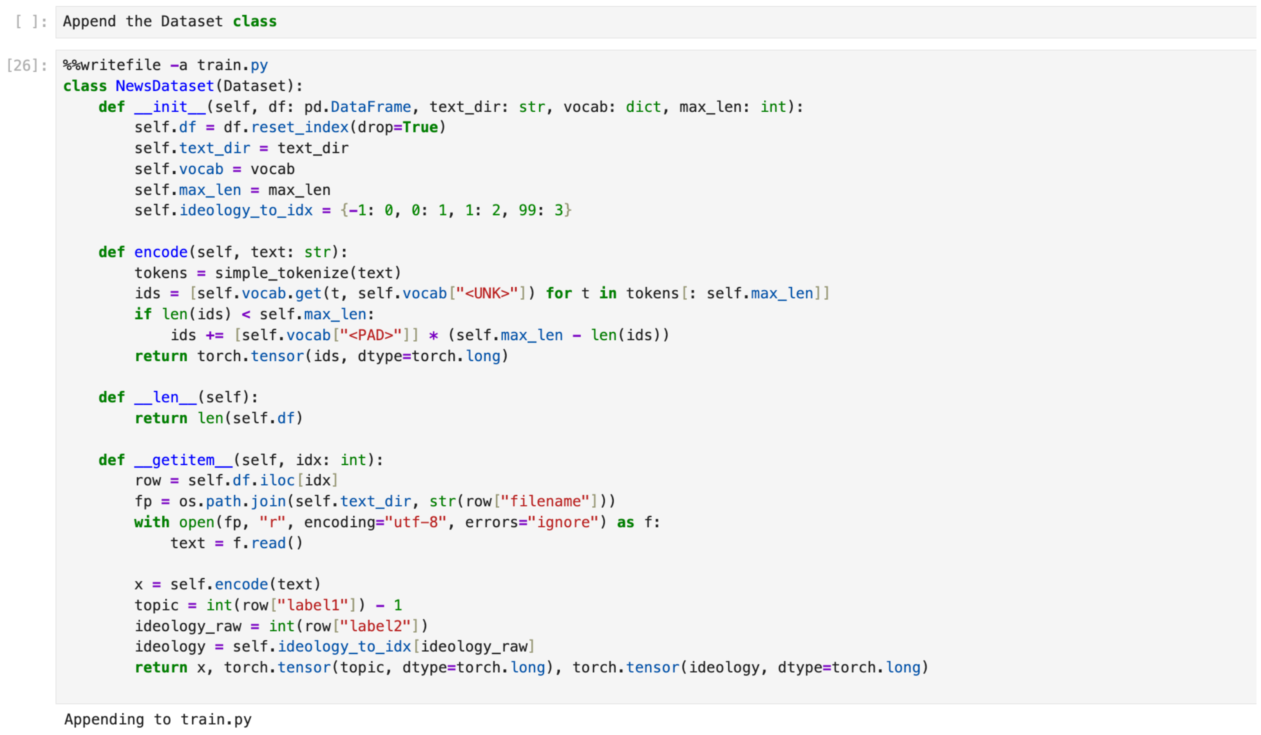
```
/Users/vincent_sichula/Desktop/Temporal Models
assignment1_Implementation of RNN and LSTM_VSichula.ipynb
assignment3-1_train_unzipped
assignment3-1_train.zip
ground_truth_train.csv
inference.py
train.py
```

Build train.py

[25]:
```
%%writefile train.py
import os
import re
import json
import argparse
from collections import Counter

import numpy as np
import pandas as pd
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader, WeightedRandomSampler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix


def seed_everything(seed: int = 42):
    import random
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed_all(seed)


def simple_tokenize(text: str):
    return re.findall(r"[A-Za-z']+", text.lower())


def compute_class_weights(y, num_classes: int):
    counts = np.bincount(y, minlength=num_classes).astype(np.float32)
    counts[counts == 0] = 1.0
    inv = 1.0 / counts
    w = inv * (num_classes / inv.sum())
    return torch.tensor(w, dtype=torch.float32)
```

```
Overwriting train.py
```

2. Dataset class

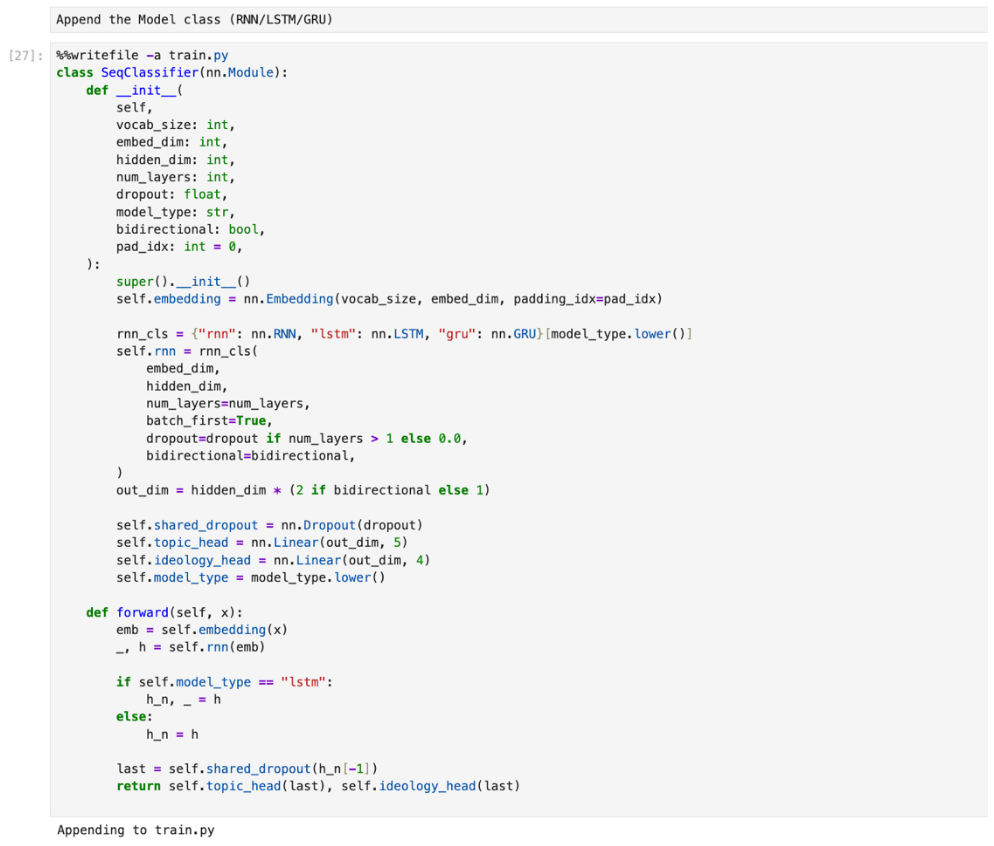


```
[ ]: Append the Dataset class
```

```
[26]: %%writefile -a train.py
class NewsDataset(Dataset):
    def __init__(self, df: pd.DataFrame, text_dir: str, vocab: dict, max_len: int):
        self.df = df.reset_index(drop=True)
        self.text_dir = text_dir
        self.vocab = vocab
        self.max_len = max_len
        self.ideology_to_idx = {-1: 0, 0: 1, 1: 2, 99: 3}

    def encode(self, text: str):
        tokens = simple_tokenize(text)
        ids = [self.vocab.get(t, self.vocab["<UNK>"]) for t in tokens[: self.max_len]]
        if len(ids) < self.max_len:
            ids += [self.vocab["<PAD>"]] * (self.max_len - len(ids))
        return torch.tensor(ids, dtype=torch.long)

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx: int):
        row = self.df.iloc[idx]
        fp = os.path.join(self.text_dir, str(row["filename"]))
        with open(fp, "r", encoding="utf-8", errors="ignore") as f:
            text = f.read()

        x = self.encode(text)
        topic = int(row["label1"]) - 1
        ideology_raw = int(row["label2"])
        ideology = self.ideology_to_idx[ideology_raw]
        return x, torch.tensor(topic, dtype=torch.long), torch.tensor(ideology, dtype=torch.long)
```

```
Appending to train.py
```

3. Model class (RNN/LSTM/GRU)

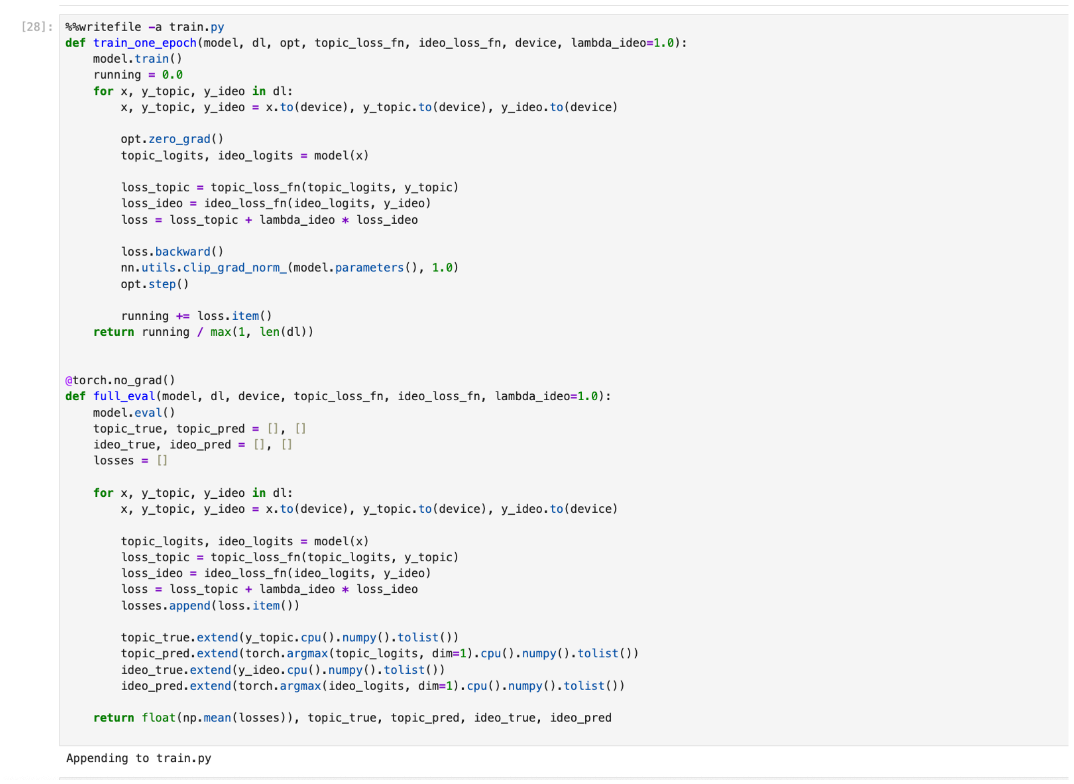


```
Append the Model class (RNN/LSTM/GRU)
```

```
[27]: %%writefile -a train.py
class SeqClassifier(nn.Module):
    def __init__(
        self,
        vocab_size: int,
        embed_dim: int,
        hidden_dim: int,
        num_layers: int,
        dropout: float,
        model_type: str,
        bidirectional: bool,
        pad_idx: int = 0,
    ):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim, padding_idx=pad_idx)

        rnn_cls = {"rnn": nn.RNN, "lstm": nn.LSTM, "gru": nn.GRU}[model_type.lower()]
        self.rnn = rnn_cls(
            embed_dim,
            hidden_dim,
            num_layers=num_layers,
            batch_first=True,
            dropout=dropout if num_layers > 1 else 0.0,
            bidirectional=bidirectional,
        )
        out_dim = hidden_dim * (2 if bidirectional else 1)

        self.shared_dropout = nn.Dropout(dropout)
        self.topic_head = nn.Linear(out_dim, 5)
        self.ideology_head = nn.Linear(out_dim, 4)
        self.model_type = model_type.lower()

    def forward(self, x):
        emb = self.embedding(x)
        _, h = self.rnn(emb)

        if self.model_type == "lstm":
            h_n, _ = h
        else:
            h_n = h

        last = self.shared_dropout(h_n[-1])
        return self.topic_head(last), self.ideology_head(last)
```

```
Appending to train.py
```

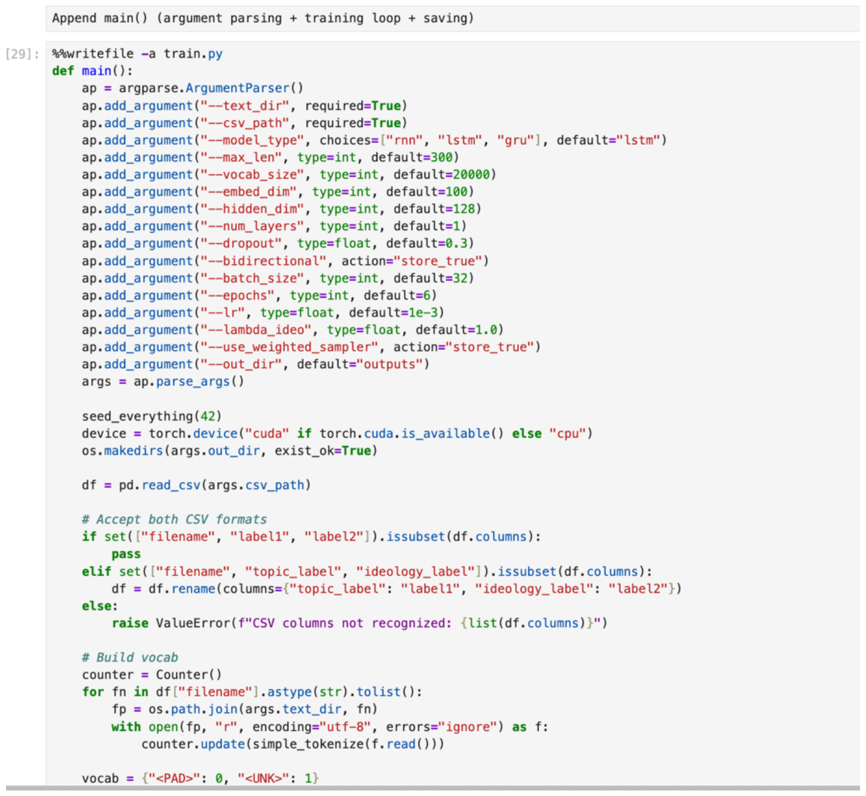
4. Training and evaluation loops



```
[28]: %%writefile -a train.py
def train_one_epoch(model, dl, opt, topic_loss_fn, ideo_loss_fn, device, lambda_ideo=1.0):
    model.train()
    running = 0.0
    for x, y_topic, y_ideo in dl:
        x, y_topic, y_ideo = x.to(device), y_topic.to(device), y_ideo.to(device)

        opt.zero_grad()
        topic_logits, ideo_logits = model(x)

        loss_topic = topic_loss_fn(topic_logits, y_topic)
        loss_ideo = ideo_loss_fn(ideo_logits, y_ideo)
        loss = loss_topic + lambda_ideo * loss_ideo

        loss.backward()
        nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        opt.step()

        running += loss.item()
    return running / max(1, len(dl))


@torch.no_grad()
def full_eval(model, dl, device, topic_loss_fn, ideo_loss_fn, lambda_ideo=1.0):
    model.eval()
    topic_true, topic_pred = [], []
    ideo_true, ideo_pred = [], []
    losses = []

    for x, y_topic, y_ideo in dl:
        x, y_topic, y_ideo = x.to(device), y_topic.to(device), y_ideo.to(device)

        topic_logits, ideo_logits = model(x)
        loss_topic = topic_loss_fn(topic_logits, y_topic)
        loss_ideo = ideo_loss_fn(ideo_logits, y_ideo)
        loss = loss_topic + lambda_ideo * loss_ideo
        losses.append(loss.item())

        topic_true.extend(y_topic.cpu().numpy().tolist())
        topic_pred.extend(torch.argmax(topic_logits, dim=1).cpu().numpy().tolist())
        ideo_true.extend(y_ideo.cpu().numpy().tolist())
        ideo_pred.extend(torch.argmax(ideo_logits, dim=1).cpu().numpy().tolist())

    return float(np.mean(losses)), topic_true, topic_pred, ideo_true, ideo_pred
```

```
Appending to train.py
```

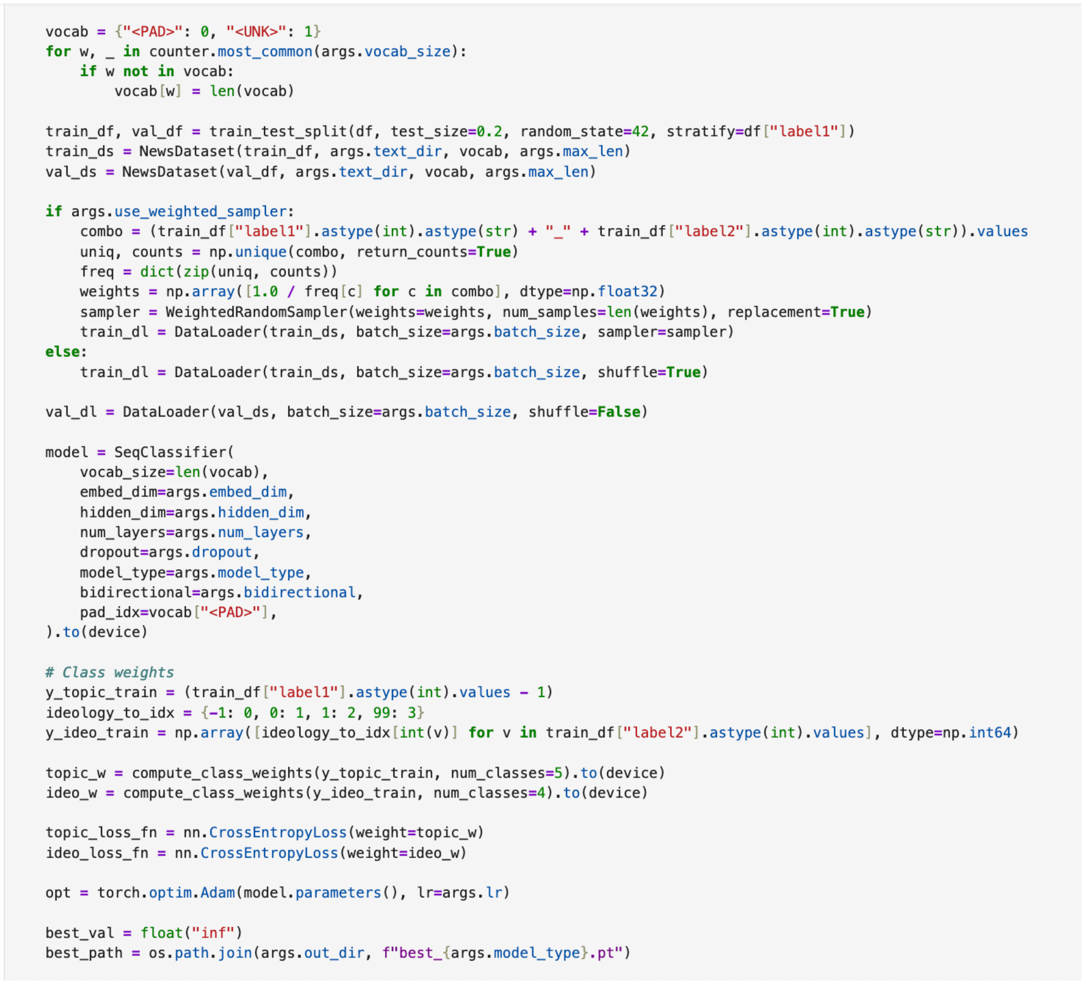
5. Main entry point (argument parsing, 80/20 split, saving outputs)



Append main() (argument parsing + training loop + saving)

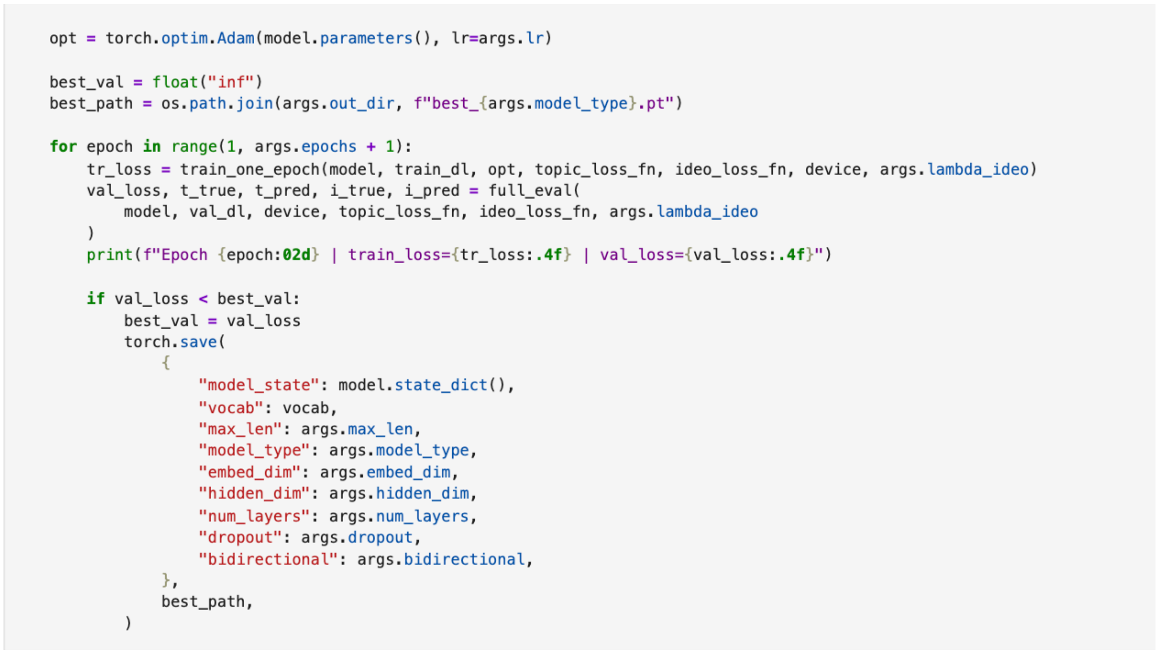
```
[29]: %%writefile -a train.py
def main():
    ap = argparse.ArgumentParser()
    ap.add_argument("--text_dir", required=True)
    ap.add_argument("--csv_path", required=True)
    ap.add_argument("--model_type", choices=["rnn", "lstm", "gru"], default="lstm")
    ap.add_argument("--max_len", type=int, default=300)
    ap.add_argument("--vocab_size", type=int, default=20000)
    ap.add_argument("--embed_dim", type=int, default=100)
    ap.add_argument("--hidden_dim", type=int, default=128)
    ap.add_argument("--num_layers", type=int, default=1)
    ap.add_argument("--dropout", type=float, default=0.3)
    ap.add_argument("--bidirectional", action="store_true")
    ap.add_argument("--batch_size", type=int, default=32)
    ap.add_argument("--epochs", type=int, default=6)
    ap.add_argument("--lr", type=float, default=1e-3)
    ap.add_argument("--lambda_ideo", type=float, default=1.0)
    ap.add_argument("--use_weighted_sampler", action="store_true")
    ap.add_argument("--out_dir", default="outputs")
    args = ap.parse_args()

    seed_everything(42)
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    os.makedirs(args.out_dir, exist_ok=True)

    df = pd.read_csv(args.csv_path)

    # Accept both CSV formats
    if set(["filename", "label1", "label2"]).issubset(df.columns):
        pass
    elif set(["filename", "topic_label", "ideology_label"]).issubset(df.columns):
        df = df.rename(columns={"topic_label": "label1", "ideology_label": "label2"})
    else:
        raise ValueError(f"CSV columns not recognized: {list(df.columns)}")

    # Build vocab
    counter = Counter()
    for fn in df["filename"].astype(str).tolist():
        fp = os.path.join(args.text_dir, fn)
        with open(fp, "r", encoding="utf-8", errors="ignore") as f:
            counter.update(simple_tokenize(f.read()))

    vocab = {"<PAD>": 0, "<UNK>": 1}
```

code continued:



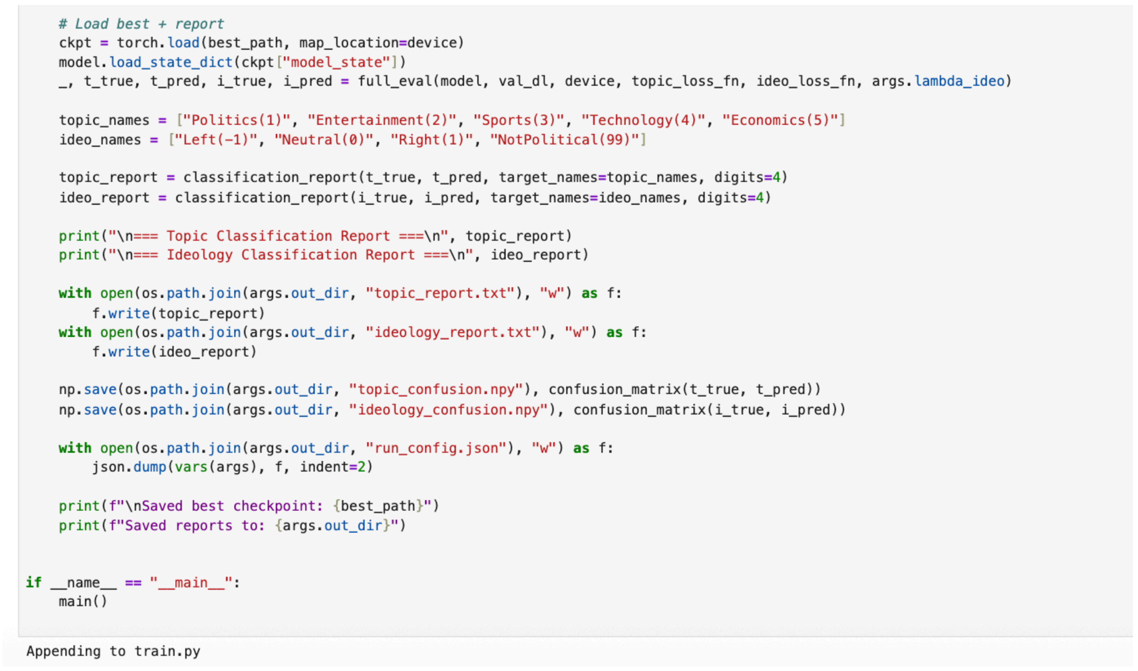
```
vocab = {"<PAD>": 0, "<UNK>": 1}
for w, _ in counter.most_common(args.vocab_size):
    if w not in vocab:
        vocab[w] = len(vocab)

train_df, val_df = train_test_split(df, test_size=0.2, random_state=42, stratify=df["label1"])
train_ds = NewsDataset(train_df, args.text_dir, vocab, args.max_len)
val_ds = NewsDataset(val_df, args.text_dir, vocab, args.max_len)

if args.use_weighted_sampler:
    combo = (train_df["label1"].astype(int).astype(str) + "_" + train_df["label2"].astype(int).astype(str)).values
    uniq, counts = np.unique(combo, return_counts=True)
    freq = dict(zip(uniq, counts))
    weights = np.array([1.0 / freq[c] for c in combo], dtype=np.float32)
    sampler = WeightedRandomSampler(weights=weights, num_samples=len(weights), replacement=True)
    train_dl = DataLoader(train_ds, batch_size=args.batch_size, sampler=sampler)
else:
    train_dl = DataLoader(train_ds, batch_size=args.batch_size, shuffle=True)

val_dl = DataLoader(val_ds, batch_size=args.batch_size, shuffle=False)

model = SeqClassifier(
    vocab_size=len(vocab),
    embed_dim=args.embed_dim,
    hidden_dim=args.hidden_dim,
    num_layers=args.num_layers,
    dropout=args.dropout,
    model_type=args.model_type,
    bidirectional=args.bidirectional,
    pad_idx=vocab["<PAD>"],
).to(device)

# Class weights
y_topic_train = (train_df["label1"].astype(int).values - 1)
ideology_to_idx = {-1: 0, 0: 1, 1: 2, 99: 3}
y_ideo_train = np.array([ideology_to_idx[int(v)] for v in train_df["label2"].astype(int).values], dtype=np.int64)

topic_w = compute_class_weights(y_topic_train, num_classes=5).to(device)
ideo_w = compute_class_weights(y_ideo_train, num_classes=4).to(device)

topic_loss_fn = nn.CrossEntropyLoss(weight=topic_w)
ideo_loss_fn = nn.CrossEntropyLoss(weight=ideo_w)

opt = torch.optim.Adam(model.parameters(), lr=args.lr)

best_val = float("inf")
best_path = os.path.join(args.out_dir, f"best_{args.model_type}.pt")
```

Code continued:



```
opt = torch.optim.Adam(model.parameters(), lr=args.lr)

best_val = float("inf")
best_path = os.path.join(args.out_dir, f"best_{args.model_type}.pt")

for epoch in range(1, args.epochs + 1):
    tr_loss = train_one_epoch(model, train_dl, opt, topic_loss_fn, ideo_loss_fn, device, args.lambda_ideo)
    val_loss, t_true, t_pred, i_true, i_pred = full_eval(
        model, val_dl, device, topic_loss_fn, ideo_loss_fn, args.lambda_ideo
    )
    print(f"Epoch {epoch:02d} | train_loss={tr_loss:.4f} | val_loss={val_loss:.4f}")

    if val_loss < best_val:
        best_val = val_loss
        torch.save(
            {
                "model_state": model.state_dict(),
                "vocab": vocab,
                "max_len": args.max_len,
                "model_type": args.model_type,
                "embed_dim": args.embed_dim,
                "hidden_dim": args.hidden_dim,
                "num_layers": args.num_layers,
                "dropout": args.dropout,
                "bidirectional": args.bidirectional,
            },
            best_path,
        )
```

Code continued:



```
    # Load best + report
    ckpt = torch.load(best_path, map_location=device)
    model.load_state_dict(ckpt["model_state"])
    _, t_true, t_pred, i_true, i_pred = full_eval(model, val_dl, device, topic_loss_fn, ideo_loss_fn, args.lambda_ideo)

    topic_names = ["Politics(1)", "Entertainment(2)", "Sports(3)", "Technology(4)", "Economics(5)"]
    ideo_names = ["Left(-1)", "Neutral(0)", "Right(1)", "NotPolitical(99)"]

    topic_report = classification_report(t_true, t_pred, target_names=topic_names, digits=4)
    ideo_report = classification_report(i_true, i_pred, target_names=ideo_names, digits=4)

    print("\n=== Topic Classification Report ===\n", topic_report)
    print("\n=== Ideology Classification Report ===\n", ideo_report)

    with open(os.path.join(args.out_dir, "topic_report.txt"), "w") as f:
        f.write(topic_report)
    with open(os.path.join(args.out_dir, "ideology_report.txt"), "w") as f:
        f.write(ideo_report)

    np.save(os.path.join(args.out_dir, "topic_confusion.npy"), confusion_matrix(t_true, t_pred))
    np.save(os.path.join(args.out_dir, "ideology_confusion.npy"), confusion_matrix(i_true, i_pred))

    with open(os.path.join(args.out_dir, "run_config.json"), "w") as f:
        json.dump(vars(args), f, indent=2)

    print(f"\nSaved best checkpoint: {best_path}")
    print(f"Saved reports to: {args.out_dir}")


if __name__ == "__main__":
    main()
```

```
Appending to train.py
```