04_lstm_sequence_model

December 14, 2025

1 LSTM Sequence Model

Objective: Train a Deep Learning sequence model (LSTM) to detect AI-generated text. Unlike the feature-based models in Notebook 3 which relied on manual counts (like sentence length), this model learns patterns directly from the raw sequence of words.

Key Components: 1. **Load Fixed Splits:** Import the pre-split data (`train.csv`, `val.csv`, `test.csv`) saved in Notebook 3 to ensure we are testing on the exact same "Collected" dataset. 2. **Preprocessing:** * **Tokenization:** Custom regex tokenizer to split text into words. * **Vocabulary:** Built *strictly* on the Training set to prevent data leakage. * **Sequence Handling:** Implements padding and "packed sequences" to handle variable-length text efficiently in PyTorch. 3. **Model Architecture:** * **Embedding Layer:** Converts words into dense vectors. * **Bi-LSTM:** Bidirectional Long Short-Term Memory layer to capture context from both past and future words. * **Classifier:** Fully connected layer with Dropout for regularization. 4. **Training Loop:** * Uses **AdamW** optimizer and **BCEWithLogitsLoss**. * Implements **Early Stopping** based on Validation F1-score to prevent overfitting. 5. **Final Evaluation:** Reports strict accuracy and F1 metrics on the held-out Test set (Scraped Data).

[16]:
```
# === LOAD FIXED SPLITS (exported from baseline notebook) ===

from google.colab import drive
drive.mount("/content/drive")

import json
from pathlib import Path
import pandas as pd
import numpy as np

ART_DIR = Path("/content/drive/MyDrive/artifacts/data_splits_v1")  # same␣
↪folder used in baseline

# --- load metadata ---
with open(ART_DIR / "meta.json") as f:
    meta = json.load(f)

fmt = meta["format"]
style_cols = meta["style_cols"]
```

```
# --- load datasets ---
if fmt == "parquet":
    train_df = pd.read_parquet(ART_DIR / "train_all.parquet")
    val_df   = pd.read_parquet(ART_DIR / "val_all.parquet")
    test_df  = pd.read_parquet(ART_DIR / "test_all.parquet")
else:
    train_df = pd.read_csv(ART_DIR / "train_all.csv")
    val_df   = pd.read_csv(ART_DIR / "val_all.csv")
    test_df  = pd.read_csv(ART_DIR / "test_all.csv")

# --- sanity checks (text + label + style columns) ---
required_cols = ["text", "label", "source"] + style_cols

for name, df in [("train", train_df), ("val", val_df), ("test", test_df)]:
    missing = [c for c in required_cols if c not in df.columns]
    if missing:
        raise ValueError(f"{name} split missing columns: {missing[:15]}{' ...' if len(missing) > 15 else ''}")

# --- labels as numpy arrays ---
y_train = train_df["label"].astype(int).values
y_val   = val_df["label"].astype(int).values
y_test  = test_df["label"].astype(int).values

print("Loaded splits from:", ART_DIR)
print("Format:", fmt)
print("Sizes:", len(train_df), len(val_df), len(test_df))
print("Label dist train:", np.bincount(y_train))
print("Label dist val:  ", np.bincount(y_val))
print("Label dist test: ", np.bincount(y_test))
print("Num stylometry features:", len(style_cols))
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
Loaded splits from: /content/drive/MyDrive/artifacts/data_splits_v1
Format: parquet
Sizes: 32615 5756 1629
Label dist train: [16677 15938]
Label dist val:   [2943 2813]
Label dist test:  [ 380 1249]
Num stylometry features: 33
```

[17]:
```
train_df
```

[17]:

	text	source	num_chars	\
0	Social Media has become a big part of our live…	kaggle	2189	
1	I am for the development OS these cars. Why, b…	kaggle	2177	

```
2      The Electoral College is a unique system in th…  kaggle       2039
3      Many of you people who believe in TLE paranorm…  kaggle       1224
4      Many schools set up summer projects to provide…  kaggle       3010
…                                                    …     …        …
32610  Thomas Jefferson was a man of many ideas and a…  kaggle       2487
32611  Dear Principal, I do think it's fair to let on…  kaggle       1791
32612  Some people agree with making students partici…  kaggle       2706
32613  There are pros and cons to learning to compete…  kaggle        785
32614  Driverless cars, also known as autonomous cars…  kaggle       3038

       num_words  num_sentences  avg_sentence_length  type_token_ratio  \
0            411             26            15.807693          0.406326
1            401             24            16.708334          0.438903
2            341             16            21.312500          0.445748
3            220             13            16.923077          0.531818
4            520             27            19.259260          0.334615
…          …             …                 …                …
32610        437             25            17.480000          0.352403
32611        357             15            23.799999          0.445378
32612        460             24            19.166666          0.300000
32613        134              6            22.333334          0.425373
32614        474             24            19.750000          0.415612

       unique_words  pct_punct  pct_upper  …  colon_ratio_emphasis  \
0               167   0.035633   0.023755  …              0.000000
1               176   0.021589   0.022049  …              0.000459
2               152   0.022070   0.020598  …              0.000000
3               117   0.024510   0.059641  …              0.000000
4               174   0.014286   0.031561  …              0.000000
…             …         …        …  …                   …
32610           154   0.015279   0.013269  …              0.000000
32611           159   0.027917   0.026242  …              0.000000
32612           138   0.015521   0.008869  …              0.000000
32613            57   0.016561   0.014013  …              0.000000
32614           197   0.019421   0.007900  …              0.000000

       semicolon_ratio_emphasis  emphasis_punctuation_ratio  \
0                      0.000000                    0.000000
1                      0.000000                    0.000459
2                      0.000000                    0.000000
3                      0.000000                    0.000000
4                      0.000332                    0.000332
…                         …                         …
32610                  0.000000                    0.000000
32611                  0.000558                    0.000558
32612                  0.000000                    0.000000
32613                  0.000000                    0.000000
```

```
32614                  0.000000                    0.000000

       hedge_phrase_count  hedge_phrase_ratio  hedge_line_ratio  \
0                       0                 0.0               0.0
1                       0                 0.0               0.0
2                       0                 0.0               0.0
3                       0                 0.0               0.0
4                       0                 0.0               0.0
…                     …                   …                 …
32610                   0                 0.0               0.0
32611                   0                 0.0               0.0
32612                   0                 0.0               0.0
32613                   0                 0.0               0.0
32614                   1                 1.0               1.0

       generic_claim_ratio  vague_quantifier_ratio  fake_citation_ratio  label
0                      0.0                     1.0                  0.0      1
1                      0.0                     0.0                  0.0      0
2                      0.0                     1.0                  0.0      1
3                      0.0                     1.0                  0.0      0
4                      0.0                     1.0                  0.0      0
…                    …                       …                    …    …
32610                  0.0                     1.0                  0.0      1
32611                  0.0                     1.0                  0.0      0
32612                  0.0                     1.0                  0.0      0
32613                  0.0                     0.0                  0.0      1
32614                  0.0                     1.0                  0.0      1

[32615 rows x 36 columns]
```

[3]:
```
# ========================
# LSTM text classifier
#   TRAIN/VAL = Kaggle only
#   TEST      = Scraped only
#
# Uses PyTorch. Includes:
# - fast regex tokenization
# - vocab built from TRAIN only
# - truncation to max_len
# - padding + packed sequences
# - early stopping on VAL F1
# - per-epoch timing
#
# datasets:
#   train_df, val_df, test_df
#   y_train, y_val, y_test
#
```

```python
# This code prints time/epoch + a simple projected total after epoch 1.
# ========================

import re
import time
import math
import numpy as np
from collections import Counter

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader

from sklearn.metrics import precision_recall_fscore_support, accuracy_score

# -------------------------
# Config (tune these first)
# -------------------------
SEED = 42
MAX_VOCAB = 50_000        # 30k-100k typical
MIN_FREQ = 2              # drop very rare tokens, since we want to focus on statistical construct of the text
MAX_LEN = 384             # 256/384/512. Higher = slower but captures longer context
BATCH_SIZE = 32           # 32/64/128 depending on CPU & RAM
EMB_DIM = 192             # 128-256
HID_DIM = 192             # 128-256
NUM_LAYERS = 1            # 1-2 (2 slower)
BIDIR = True
DROPOUT = 0.2
LR = 2e-3
EPOCHS = 8
PATIENCE = 4              # early stop if VAL F1 doesn't improve
CLIP = 1.0

DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", DEVICE)

# -------------------------
# Reproducibility
# -------------------------
torch.manual_seed(SEED)
np.random.seed(SEED)

# -------------------------
# Tokenizer (fast, stable)
# -------------------------
```

```
_tok = re.compile(r"[A-Za-z0-9]+(?:'[A-Za-z0-9]+)?|[^\sA-Za-z0-9]")
def tokenize(text: str):
    return _tok.findall((text or "").lower())

# -------------------------
# Build vocab from TRAIN only
# -------------------------
def build_vocab(texts, max_vocab=MAX_VOCAB, min_freq=MIN_FREQ):
    counter = Counter()
    for t in texts:
        counter.update(tokenize(t))
    # Special tokens
    itos = ["<pad>", "<unk>"]
    # Keep most common above min_freq
    for tok, freq in counter.most_common():
        if freq < min_freq:
            break
        itos.append(tok)
        if len(itos) >= max_vocab:
            break
    stoi = {tok: i for i, tok in enumerate(itos)}
    return stoi, itos

train_texts = train_df["text"].astype(str).tolist()
val_texts   = val_df["text"].astype(str).tolist()
test_texts  = test_df["text"].astype(str).tolist()

stoi, itos = build_vocab(train_texts)
PAD_IDX = stoi["<pad>"]
UNK_IDX = stoi["<unk>"]

print(f"Vocab size: {len(itos):,} (PAD={PAD_IDX}, UNK={UNK_IDX})")

# -------------------------
# Dataset + Collate
# -------------------------
def encode(text: str, max_len=MAX_LEN):
    ids = [stoi.get(tok, UNK_IDX) for tok in tokenize(text)]
    if len(ids) > max_len:
        ids = ids[:max_len]
    return ids

class TextDataset(Dataset):
    def __init__(self, texts, labels):
        self.texts = texts
        self.labels = labels.astype(np.int64)
    def __len__(self):
```

```
        return len(self.texts)
    def __getitem__(self, i):
        return self.texts[i], self.labels[i]

def collate_batch(batch):
    texts, labels = zip(*batch)
    seqs = [torch.tensor(encode(t), dtype=torch.long) for t in texts]
    lengths = torch.tensor([len(s) for s in seqs], dtype=torch.long)

    # pad to max length in batch
    maxl = int(lengths.max().item()) if len(lengths) else 1
    padded = torch.full((len(seqs), maxl), PAD_IDX, dtype=torch.long)
    for i, s in enumerate(seqs):
        padded[i, :len(s)] = s

    labels = torch.tensor(labels, dtype=torch.float32)  # binary
    return padded.to(DEVICE), lengths.to(DEVICE), labels.to(DEVICE)

train_ds = TextDataset(train_texts, y_train)
val_ds   = TextDataset(val_texts, y_val)
test_ds  = TextDataset(test_texts, y_test)

train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True, ␣
 ↪collate_fn=collate_batch)
val_loader   = DataLoader(val_ds,   batch_size=BATCH_SIZE, shuffle=False,␣
 ↪collate_fn=collate_batch)
test_loader  = DataLoader(test_ds,  batch_size=BATCH_SIZE, shuffle=False,␣
 ↪collate_fn=collate_batch)
```

```
Using device: cuda
Vocab size: 36,117 (PAD=0, UNK=1)
```

[4]:
```
# Model
class LSTMClassifier(nn.Module):
    def __init__(self, vocab_size, emb_dim, hid_dim, num_layers, bidir,␣
 ↪dropout, pad_idx):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, emb_dim, padding_idx=pad_idx)
        self.lstm = nn.LSTM(
            input_size=emb_dim,
            hidden_size=hid_dim,
            num_layers=num_layers,
            batch_first=True,
            bidirectional=bidir,
            dropout=0.0 if num_layers == 1 else dropout
        )
        out_dim = hid_dim * (2 if bidir else 1)
```

```
        self.dropout = nn.Dropout(dropout)
        self.fc = nn.Linear(out_dim, 1)

    def forward(self, x, lengths):
        # x: [B, T]
        emb = self.dropout(self.embedding(x))  # [B, T, E]

        # pack sequences for speed
        lengths_cpu = lengths.to("cpu")
        packed = nn.utils.rnn.pack_padded_sequence(emb, lengths_cpu,␣
 ↪batch_first=True, enforce_sorted=False)
        packed_out, (h, c) = self.lstm(packed)

        # h shape: [num_layers * num_directions, B, H]
        if self.lstm.bidirectional:
            # last layer forward + backward
            h_f = h[-2, :, :]
            h_b = h[-1, :, :]
            h_cat = torch.cat([h_f, h_b], dim=1)  # [B, 2H]
        else:
            h_cat = h[-1, :, :]  # [B, H]

        logits = self.fc(self.dropout(h_cat)).squeeze(1)  # [B]
        return logits

model = LSTMClassifier(
    vocab_size=len(itos),
    emb_dim=EMB_DIM,
    hid_dim=HID_DIM,
    num_layers=NUM_LAYERS,
    bidir=BIDIR,
    dropout=DROPOUT,
    pad_idx=PAD_IDX
).to(DEVICE)

criterion = nn.BCEWithLogitsLoss()
optimizer = torch.optim.AdamW(model.parameters(), lr=LR)
```

[5]:
```
# -------------------------
# Metrics helpers
# -------------------------
@torch.no_grad()
def predict_loader(loader):
    model.eval()
    all_probs, all_y = [], []
    for x, lengths, y in loader:
        logits = model(x, lengths)
```

```
        probs = torch.sigmoid(logits).detach().cpu().numpy()
        all_probs.append(probs)
        all_y.append(y.detach().cpu().numpy())
    probs = np.concatenate(all_probs) if all_probs else np.array([])
    ytrue = np.concatenate(all_y).astype(int) if all_y else np.array([],␣
 ↪dtype=int)
    return probs, ytrue

def eval_split(loader, threshold=0.5):
    probs, ytrue = predict_loader(loader)
    ypred = (probs >= threshold).astype(int)
    acc = accuracy_score(ytrue, ypred)
    p, r, f1, _ = precision_recall_fscore_support(ytrue, ypred,␣
 ↪average="binary", zero_division=0)
    return acc, p, r, f1
```

[6]:
```
# -------------------------
# Train loop with timing + early stopping
# -------------------------
best_val_f1 = -1.0
best_state = None
no_improve = 0
epoch_times = []

for epoch in range(1, EPOCHS + 1):
    train_loss_hist = []
    val_f1_hist = []
    val_acc_hist = []
    t0 = time.time()
    model.train()
    running_loss = 0.0
    n_batches = 0

    for x, lengths, y in train_loader:
        optimizer.zero_grad()
        logits = model(x, lengths)
        loss = criterion(logits, y)
        loss.backward()
        nn.utils.clip_grad_norm_(model.parameters(), CLIP)
        optimizer.step()

        running_loss += float(loss.item())
        n_batches += 1

    train_loss = running_loss / max(n_batches, 1)
    val_acc, val_p, val_r, val_f1 = eval_split(val_loader)
```

```
    t1 = time.time()
    epoch_sec = t1 - t0
    epoch_times.append(epoch_sec)
    train_loss_hist.append(train_loss)
    val_f1_hist.append(val_f1)
    val_acc_hist.append(val_acc)


    # After epoch 1, print a rough projection based on observed time/epoch
    if epoch == 1:
        projected = epoch_sec * EPOCHS
        print(f"\n[Timing] Epoch 1 took {epoch_sec:.1f}s. Rough projected total␣
↪for {EPOCHS} epochs: ~{projected/60:.1f} min (before early stopping).")

    print(f"Epoch {epoch}/{EPOCHS} | train_loss={train_loss:.4f} | "
          f"VAL acc={val_acc:.4f} p={val_p:.4f} r={val_r:.4f} f1={val_f1:.4f} |␣
↪"
          f"time={epoch_sec:.1f}s")

    # Early stopping on VAL F1
    if val_f1 > best_val_f1 + 1e-4:
        best_val_f1 = val_f1
        best_state = {k: v.cpu().clone() for k, v in model.state_dict().items()}
        no_improve = 0
    else:
        no_improve += 1
        if no_improve >= PATIENCE:
            print(f"Early stopping: no VAL F1 improvement for {PATIENCE}␣
↪epoch(s).")
            break

# Restore best model
if best_state is not None:
    model.load_state_dict(best_state)
```

```
[Timing] Epoch 1 took 40.0s. Rough projected total for 8 epochs: ~5.3 min
(before early stopping).
Epoch 1/8 | train_loss=0.1840 | VAL acc=0.9715 p=0.9675 r=0.9744 f1=0.9710 |
time=40.0s
Epoch 2/8 | train_loss=0.0566 | VAL acc=0.9842 p=0.9931 r=0.9744 f1=0.9837 |
time=37.8s
Epoch 3/8 | train_loss=0.0293 | VAL acc=0.9858 p=0.9956 r=0.9751 f1=0.9853 |
time=37.6s
Epoch 4/8 | train_loss=0.0161 | VAL acc=0.9906 p=0.9922 r=0.9886 f1=0.9904 |
time=37.5s
Epoch 5/8 | train_loss=0.0084 | VAL acc=0.9851 p=0.9985 r=0.9708 f1=0.9845 |
```

```
time=37.4s
Epoch 6/8 | train_loss=0.0092 | VAL acc=0.9913 p=0.9953 r=0.9868 f1=0.9911 |
time=37.4s
Epoch 7/8 | train_loss=0.0045 | VAL acc=0.9913 p=0.9936 r=0.9886 f1=0.9911 |
time=37.5s
Epoch 8/8 | train_loss=0.0039 | VAL acc=0.9918 p=0.9925 r=0.9908 f1=0.9916 |
time=37.6s
```

[7]:
```python
# -------------------------
# Final evaluation
# -------------------------
val_acc, val_p, val_r, val_f1 = eval_split(val_loader)
test_acc, test_p, test_r, test_f1 = eval_split(test_loader)

print("\n===== FINAL (best checkpoint) =====")
print(f"VAL  acc={val_acc:.4f} p={val_p:.4f} r={val_r:.4f} f1={val_f1:.4f}")
print(f"TEST acc={test_acc:.4f} p={test_p:.4f} r={test_r:.4f} f1={test_f1:.4f}")
print(f"Avg time/epoch: {np.mean(epoch_times):.1f}s over {len(epoch_times)}␣
 ↪epoch(s)")
```

```
===== FINAL (best checkpoint) =====
VAL  acc=0.9918 p=0.9925 r=0.9908 f1=0.9916
TEST acc=0.7109 p=0.8016 r=0.8279 f1=0.8145
Avg time/epoch: 37.8s over 8 epoch(s)
```

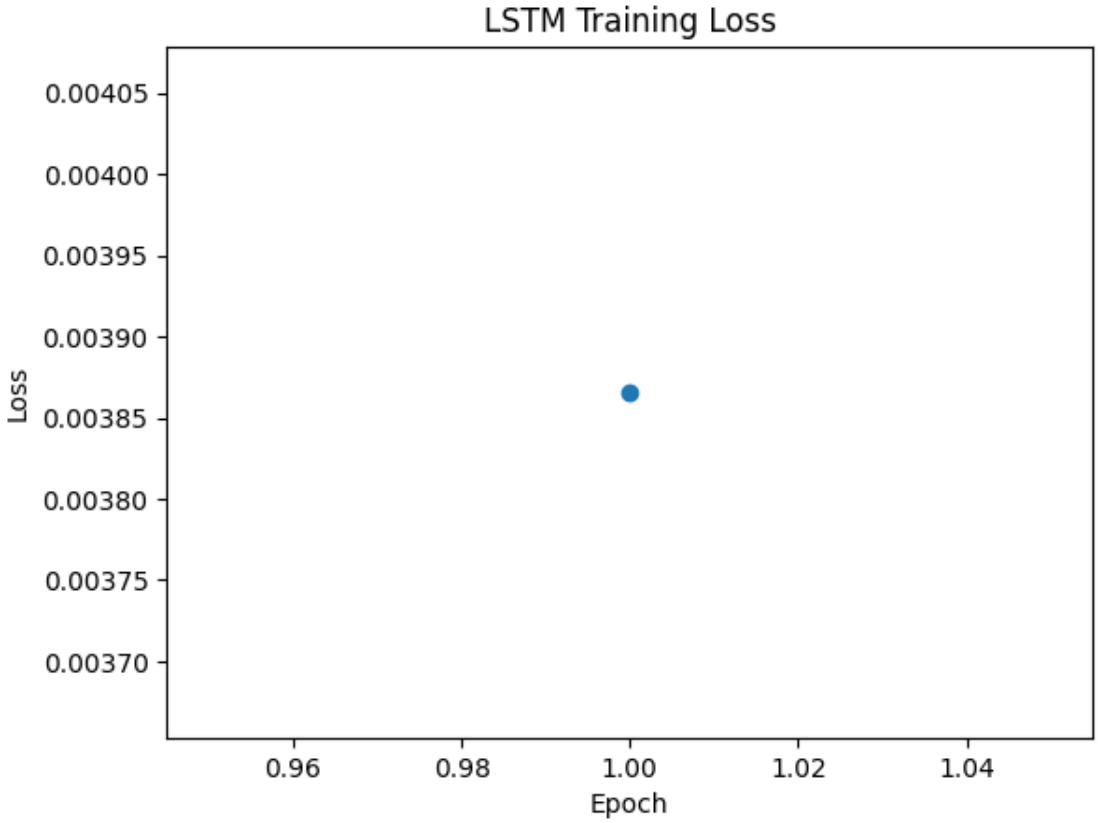
[9]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.metrics import (
    confusion_matrix, ConfusionMatrixDisplay,
    precision_recall_fscore_support, accuracy_score,
    roc_auc_score, average_precision_score
)

# -------------------------
# 1) Evaluate on a loader (return probs + labels)
# -------------------------
@torch.no_grad()
def lstm_probs_on_loader(loader):
    model.eval()
    all_probs, all_y = [], []
    for x, lengths, y in loader:
        logits = model(x, lengths)
        probs = torch.sigmoid(logits).detach().cpu().numpy()
        all_probs.append(probs)
```

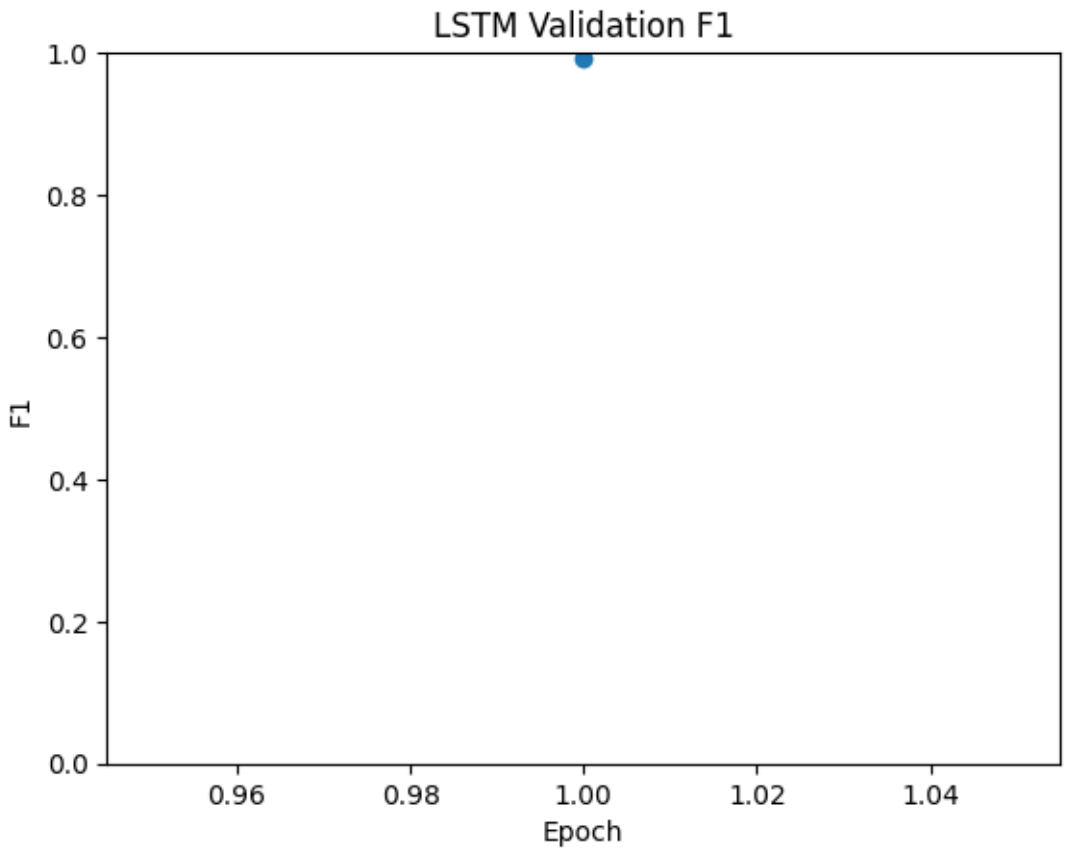
```
        all_y.append(y.detach().cpu().numpy())
    probs = np.concatenate(all_probs) if all_probs else np.array([])
    ytrue = np.concatenate(all_y).astype(int) if all_y else np.array([],␣
 ↪dtype=int)
    return probs, ytrue

def metrics_from_probs(ytrue, probs, thr=0.5):
    ypred = (probs >= thr).astype(int)
    acc = accuracy_score(ytrue, ypred)
    p, r, f1, _ = precision_recall_fscore_support(ytrue, ypred,␣
 ↪average="binary", zero_division=0)

    out = {"acc": acc, "prec": p, "rec": r, "f1": f1}
    try: out["roc_auc"] = roc_auc_score(ytrue, probs)
    except Exception: out["roc_auc"] = np.nan
    try: out["pr_auc"]  = average_precision_score(ytrue, probs)
    except Exception: out["pr_auc"]  = np.nan
    return out, ypred

# -------------------------
# 2) Plots
# -------------------------
def plot_training_curves(train_loss_hist, val_f1_hist):
    epochs = np.arange(1, len(train_loss_hist) + 1)

    fig, ax = plt.subplots()
    ax.plot(epochs, train_loss_hist, marker="o")
    ax.set_title("LSTM Training Loss")
    ax.set_xlabel("Epoch")
    ax.set_ylabel("Loss")
    plt.show()

    fig, ax = plt.subplots()
    ax.plot(epochs, val_f1_hist, marker="o")
    ax.set_title("LSTM Validation F1")
    ax.set_xlabel("Epoch")
    ax.set_ylabel("F1")
    ax.set_ylim(0, 1)
    plt.show()

def plot_confidence_distributions(ytrue, probs, marker_prob=None):
    p_h = probs[ytrue == 0]
    p_a = probs[ytrue == 1]
    bins = np.linspace(0, 1, 51)

    fig, ax = plt.subplots()
    ax.hist(p_h, bins=bins, alpha=0.6, density=True, label="True Human (0)")
```

```
    ax.hist(p_a, bins=bins, alpha=0.6, density=True, label="True AI (1)")
    if marker_prob is not None:
        ax.axvline(marker_prob, linestyle="--", linewidth=2)
        ax.text(marker_prob, ax.get_ylim()[1]*0.95, "Custom input",␣
↪rotation=90, va="top")
    ax.set_title("LSTM Prob(AI) distributions (SCRAPED TEST)")
    ax.set_xlabel("Prob(AI)")
    ax.set_ylabel("Density")
    ax.legend()
    plt.show()

def plot_length_bin_f1(test_df, ytrue, probs, thr=0.5):
    lens = test_df["text"].astype(str).str.len()
    q1, q2 = lens.quantile([0.33, 0.66]).values
    bins = pd.cut(lens, [-np.inf, q1, q2, np.inf],␣
↪labels=["short","medium","long"]).astype(str)

    ypred = (probs >= thr).astype(int)

    labels = ["short","medium","long"]
    f1s, counts = [], []
    for b in labels:
        m = (bins == b).values
        counts.append(int(m.sum()))
        if m.sum() == 0:
            f1s.append(np.nan)
        else:
            _, _, f1, _ = precision_recall_fscore_support(ytrue[m], ypred[m],␣
↪average="binary", zero_division=0)
            f1s.append(float(f1))

    fig, ax = plt.subplots()
    ax.bar(labels, f1s)
    ax.set_ylim(0, 1)
    ax.set_ylabel("F1")
    ax.set_title("LSTM F1 by Length Bin (SCRAPED TEST)")
    for i, c in enumerate(counts):
        ax.text(i, 0.02, f"n={c}", ha="center", va="bottom")
    plt.show()

def per_source_table(test_df, ytrue, probs, thr=0.5, min_n=20):
    ypred = (probs >= thr).astype(int)
    rows = []
    for src, idx in test_df.groupby("source").indices.items():
        idx = np.array(list(idx))
        if len(idx) < min_n:
            continue
```

```
        m, _ = metrics_from_probs(ytrue[idx], probs[idx], thr=thr)
        rows.append({"source": src, "n": len(idx), **m})
    out = pd.DataFrame(rows).sort_values("f1", ascending=False)
    print(f"\n=== LSTM per-source performance (SCRAPED TEST, min_n={min_n})␣
 ↪===")
    display(out.round(4))
    return out

def show_confident_errors(test_df, ytrue, probs, topk=10):
    df_err = test_df.copy().reset_index(drop=True)
    df_err["y_true"] = ytrue
    df_err["prob_ai"] = probs
    df_err["y_pred"] = (probs >= 0.5).astype(int)
    df_err["correct"] = (df_err["y_true"] == df_err["y_pred"])

    fp = df_err[(df_err["y_true"] == 0) & (df_err["y_pred"] == 1)].
 ↪sort_values("prob_ai", ascending=False).head(topk)
    fn = df_err[(df_err["y_true"] == 1) & (df_err["y_pred"] == 0)].
 ↪sort_values("prob_ai", ascending=True).head(topk)

    print(f"\n=== Top {topk} confident FALSE POSITIVES (Human predicted AI)␣
 ↪===")
    display(fp[["source","prob_ai","text"]].assign(text=fp["text"].str.slice(0,␣
 ↪300) + "..."))

    print(f"\n=== Top {topk} confident FALSE NEGATIVES (AI predicted Human)␣
 ↪===")
    display(fn[["source","prob_ai","text"]].assign(text=fn["text"].str.slice(0,␣
 ↪300) + "..."))

# -------------------------
# 3) Run dashboard
# -------------------------
# training curves (only if you logged these in Cell 1)
if "train_loss_hist" in globals() and "val_f1_hist" in globals():
    plot_training_curves(train_loss_hist, val_f1_hist)
else:
    print("No training curves found. Add train_loss_hist / val_f1_hist logging␣
 ↪as suggested.")

# evaluate val + test
val_probs, val_ytrue = lstm_probs_on_loader(val_loader)
test_probs, test_ytrue = lstm_probs_on_loader(test_loader)

val_m, _ = metrics_from_probs(val_ytrue, val_probs, thr=0.5)
test_m, test_pred = metrics_from_probs(test_ytrue, test_probs, thr=0.5)
```

```
print("\n=== LSTM Compact Metrics (thr=0.5) ===")
display(pd.DataFrame([
    {"split":"VAL",  **val_m},
    {"split":"TEST", **test_m},
]).round(4))
```

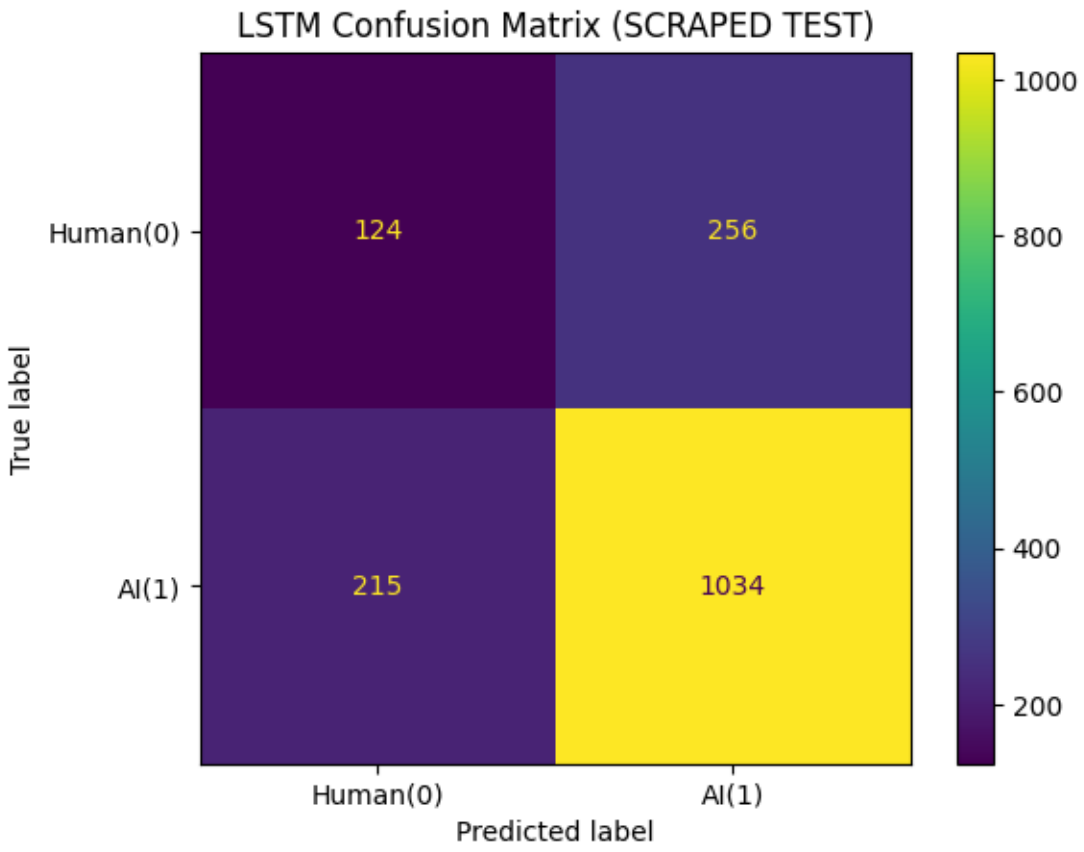




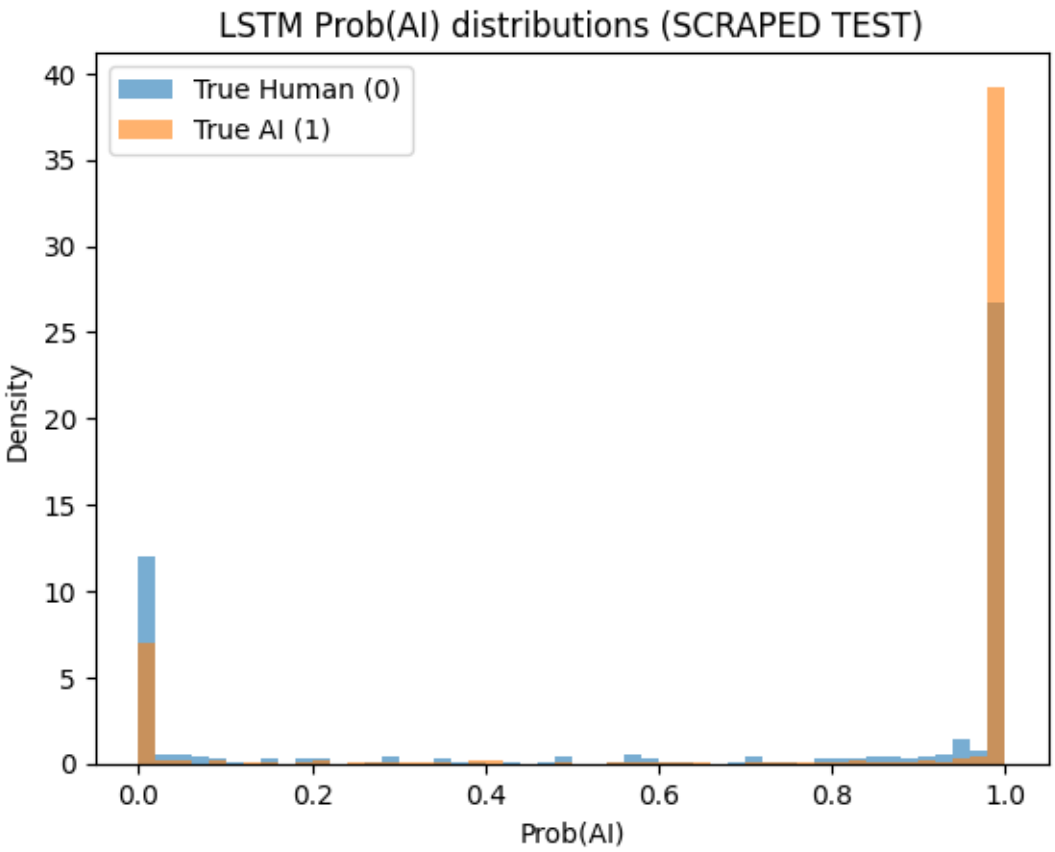

```
=== LSTM Compact Metrics (thr=0.5) ===
```

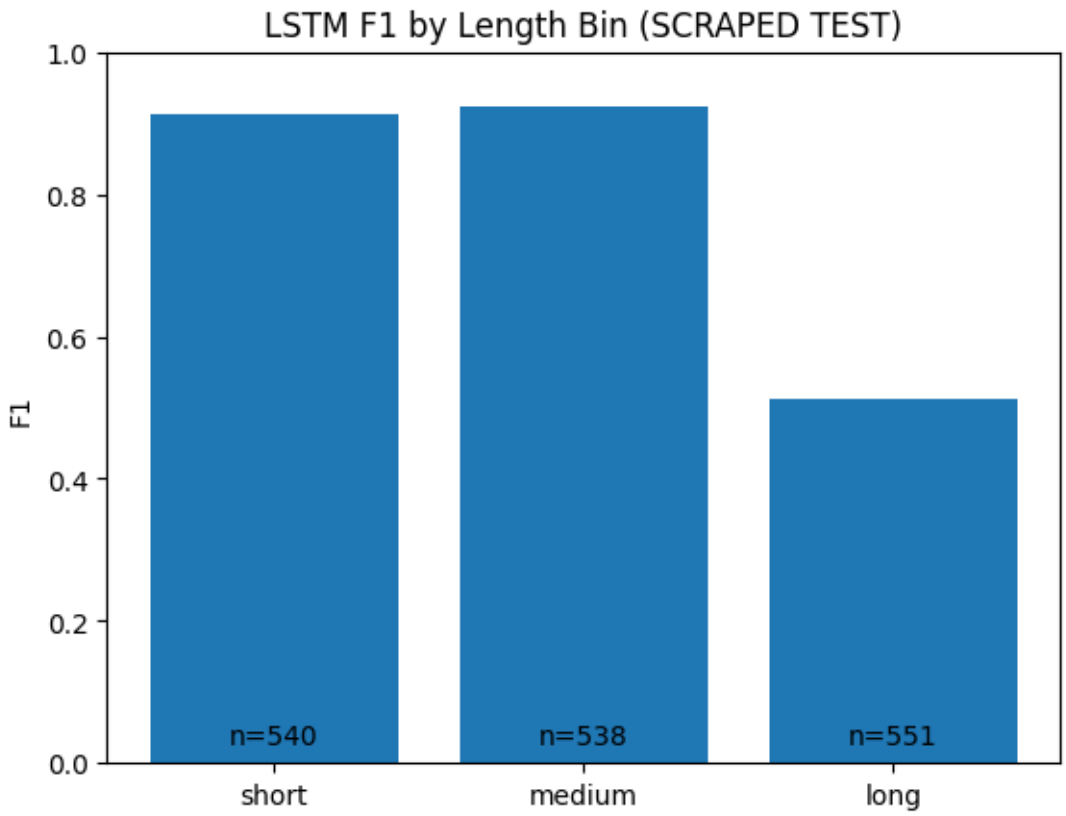
	split	acc	prec	rec	f1	roc_auc	pr_auc
0	VAL	0.9918	0.9925	0.9908	0.9916	0.9994	0.9995
1	TEST	0.7109	0.8016	0.8279	0.8145	0.7460	0.9137

[10]:
```
# confusion matrix on scraped test
cm = confusion_matrix(test_ytrue, test_pred, labels=[0,1])
disp = ConfusionMatrixDisplay(cm, display_labels=["Human(0)","AI(1)"])
fig, ax = plt.subplots()
disp.plot(ax=ax, values_format="d")
ax.set_title("LSTM Confusion Matrix (SCRAPED TEST)")
plt.show()
```



[11]:
```
# confidence distributions
plot_confidence_distributions(test_ytrue, test_probs)

# length-bin performance
plot_length_bin_f1(test_df.reset_index(drop=True), test_ytrue, test_probs,␣
↪thr=0.5)
```


LSTM Prob(AI) distributions (SCRAPED TEST)
True Human (0)
True AI (1)
Density
Prob(AI)
0
5
10
15
20
25
30
35
40
0.0
0.2
0.4
0.6
0.8
1.0



[18]:
```
# per-source
per_source_table(test_df.reset_index(drop=True), test_ytrue, test_probs, thr=0.
↪5, min_n=20)
```

```
=== LSTM per-source performance (SCRAPED TEST, min_n=20) ===

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:379:
UndefinedMetricWarning: Only one class is present in y_true. ROC AUC score is
not defined in that case.
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1033:
UserWarning: No positive class found in y_true, recall is set to one for all
thresholds.
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:379:
UndefinedMetricWarning: Only one class is present in y_true. ROC AUC score is
not defined in that case.
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:379:
UndefinedMetricWarning: Only one class is present in y_true. ROC AUC score is
not defined in that case.
```

```
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1033:
UserWarning: No positive class found in y_true, recall is set to one for all
thresholds.
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:379:
UndefinedMetricWarning: Only one class is present in y_true. ROC AUC score is
not defined in that case.
  warnings.warn(
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_ranking.py:1033:
UserWarning: No positive class found in y_true, recall is set to one for all
thresholds.
  warnings.warn(
```

	source	n	acc	prec	rec	f1	roc_auc	pr_auc
1	chatgpt	1249	0.8279	1.0	0.8279	0.9058	NaN	1.0
0	arxiv	62	0.0806	0.0	0.0000	0.0000	NaN	0.0
2	news	97	0.4948	0.0	0.0000	0.0000	NaN	0.0
3	wikipedia	221	0.3213	0.0	0.0000	0.0000	NaN	0.0

[18]:

	source	n	acc	prec	rec	f1	roc_auc	pr_auc
1	chatgpt	1249	0.827862	1.0	0.827862	0.905826	NaN	1.0
0	arxiv	62	0.080645	0.0	0.000000	0.000000	NaN	0.0
2	news	97	0.494845	0.0	0.000000	0.000000	NaN	0.0
3	wikipedia	221	0.321267	0.0	0.000000	0.000000	NaN	0.0

[19]:
```
# confident errors
show_confident_errors(test_df.reset_index(drop=True), test_ytrue, test_probs,␣
 ↪topk=10)

# expose test_probs for custom input overlay later
LSTM_TEST_PROBS = test_probs
LSTM_TEST_YTRUE = test_ytrue
```

```
=== Top 10 confident FALSE POSITIVES (Human predicted AI) ===
```

	source	prob_ai	text
115	arxiv	1.0	An Information Theoretic approach to Post Rand…
350	arxiv	1.0	Ray dynamics of whispering-gallery modes in cu…
831	arxiv	1.0	Unsupervised Learning of Solutions to Differen…
643	arxiv	1.0	Computing Valid p-values for Image Segmentatio…
692	wikipedia	1.0	late 20th and early 21st centuries has led to …
776	wikipedia	1.0	four main ethical dimensions, defined as follo…
1075	arxiv	1.0	Conditional Independence Beyond Domain Separab…
935	arxiv	1.0	Estimation of seasonal long-memory parameters…
1309	arxiv	1.0	we provide long overlooked evidence that (3) e…
1378	wikipedia	1.0	constant rates without the fluctuations in une…

```
=== Top 10 confident FALSE NEGATIVES (AI predicted Human) ===

       source       prob_ai                                                  text
398    chatgpt  2.296798e-08  Train min/max/avg: {tr_min} / {tr_max} / {tr_a…
213    chatgpt  2.696999e-08  filter={ "$and": [ {"payer_id": payer_id.lower…
668    chatgpt  3.472035e-08  using the same function as training feats = co…
1000   chatgpt  4.067079e-08  what the agent prioritizes/filters) agent_urls…
1221   chatgpt  5.867082e-08  scripts/<set>/__main__.py candidate = cur / "s…
453    chatgpt  6.216964e-08  == "O": prev_lab = "O"; continue pref, _, lab …
1611   chatgpt  8.376617e-08  / image_features.norm(dim=-1, keepdim=True) si…
427    chatgpt  8.626050e-08  advances to Step 2) # ========================…
1144   chatgpt  9.531081e-08  || r.rxn_code as oac_code, 'PH-' || r.rxn_code…
1240   chatgpt  1.005773e-07  Step 3.") st.rerun() if back_to_1: st.session_…
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[20]:
```python
# from https://gist.github.com/jonathanagustin/b67b97ef12c53a8dec27b343dca4abba
# install can take a minute

import os
# @title Convert Notebook to PDF. Save Notebook to given directory
NOTEBOOKS_DIR = "/content/drive/MyDrive/" # @param {type:"string"}
NOTEBOOK_NAME = "04_lstm_sequence_model.ipynb" # @param {type:"string"}
#-------------------------------------------------------------------------#
from google.colab import drive
drive.mount("/content/drive/", force_remount=True)
NOTEBOOK_PATH = f"{NOTEBOOKS_DIR}/{NOTEBOOK_NAME}"
assert os.path.exists(NOTEBOOK_PATH), f"NOTEBOOK NOT FOUND: {NOTEBOOK_PATH}"
```

```
!apt install -y texlive-xetex texlive-fonts-recommended texlive-plain-generic >␣
↪/dev/null 2>&1
!apt install pandoc > /dev/null 2>&1
!jupyter nbconvert "$NOTEBOOK_PATH" --to pdf > /dev/null 2>&1
NOTEBOOK_PDF = NOTEBOOK_PATH.rsplit('.', 1)[0] + '.pdf'
assert os.path.exists(NOTEBOOK_PDF), f"ERROR MAKING PDF: {NOTEBOOK_PDF}"
print(f"PDF CREATED: {NOTEBOOK_PDF}")
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/tmp/ipython-input-4169629341.py in <cell line: 0>()
      8␣
↪#-------------------------------------------------------------------------------#
      9 from google.colab import drive
---> 10 drive.mount("/content/drive/", force_remount=True)
     11 NOTEBOOK_PATH = f"{NOTEBOOKS_DIR}/{NOTEBOOK_NAME}"
     12 assert os.path.exists(NOTEBOOK_PATH), f"NOTEBOOK NOT FOUND:␣
↪{NOTEBOOK_PATH}"

/usr/local/lib/python3.12/dist-packages/google/colab/drive.py in␣
↪mount(mountpoint, force_remount, timeout_ms, readonly)
     95 def mount(mountpoint, force_remount=False, timeout_ms=120000,␣
↪readonly=False):
     96   """Mount your Google Drive at the specified mountpoint path."""
---> 97   return _mount(

     98       mountpoint,
     99       force_remount=force_remount,

/usr/local/lib/python3.12/dist-packages/google/colab/drive.py in␣
↪_mount(mountpoint, force_remount, timeout_ms, ephemeral, readonly)
    250
    251   while True:
--> 252     case = d.expect([

    253         success,
    254         prompt,

/usr/local/lib/python3.12/dist-packages/pexpect/spawnbase.py in expect(self,␣
↪pattern, timeout, searchwindowsize, async_, **kw)
    352
    353         compiled_pattern_list = self.compile_pattern_list(pattern)
--> 354         return self.expect_list(compiled_pattern_list,

    355                 timeout, searchwindowsize, async_)
    356

/usr/local/lib/python3.12/dist-packages/pexpect/spawnbase.py in␣
↪expect_list(self, pattern_list, timeout, searchwindowsize, async_, **kw)
```

```
    381             return expect_async(exp, timeout)
    382         else:
--> 383             return exp.expect_loop(timeout)
    384
    385     def expect_exact(self, pattern_list, timeout=-1, searchwindowsize=- ,

/usr/local/lib/python3.12/dist-packages/pexpect/expect.py in expect_loop(self,␣
 ↪timeout)
    169                 incoming = spawn.read_nonblocking(spawn.maxread, timeou )
    170                 if self.spawn.delayafterread is not None:
--> 171                     time.sleep(self.spawn.delayafterread)
    172                 idx = self.new_data(incoming)
    173                 # Keep reading until exception or return.

KeyboardInterrupt:
```