Temporal Modeling in Multimodal Depression Detection: A Critical Review and Cross-Platform Generalization Study

Dr. Mohsin Bilal, Alshikah Alqahtani, Shahad Almubki, Shahad Alzoman

Department of Data Sciences, College of Computer Engineering and Sciences, Prince Sattam bin Abdulaziz University, Al-Kharj, Saudi Arabia

Email: m.farooq@psau.edu.sa

Abstract

This paper assesses the work "It's Just a Matter of Time: Detecting Depression with Time-Enriched Multimodal Transformers" [1] that focuses on the Transformer-based model that incorporates text, photo, and time data to discover depression on the social media. The objective includes discussion of the design choices to the model and a subsequent experiment. The work explores whether the ability of social media time-based analysis models can be generalized to other social media platforms. We show that the use of temporal embedding data is a critical component for predicting depression on social media at a given time and is robust in cross-domain analysis, clarifying the need for incorporating data on the user's behavior over time.

I. INTRODUCTION

Given the vast volumes of generated content by users, depression detection on social media has become a hot topic of research [2]. The paper under review proposes the first-ever model of users' timelines with a multimodal transformer architecture that utilizes time2vec embeddings [3]. This work stands out for conceptualizing depression as a longitudinal phenomenon, which is consistent with its clinical understanding. Still, the hypothesis that one's digital behavior depicts the true state of one's mind is not necessarily true, and the extrapolation of these models on various platforms and cultures poses a question mark.

II. CRITICAL REVIEW

A. Strengths

- The integration of text, image, and temporal currents via transformer backbone is appropriate for the analysis of irregular, multifaceted data.
- Time2vec uses embeddings to decode the temporal gaps between posts and extract the behavioral rhythms.
- Cross-platform assessment (Twitter and Reddit) is beneficial for ecological validity.
- Comparatively high F1-score is reported, standing on the shoulders of prior work.
- Utilization of Integrated Gradients for transparency of model behavior is commendable.
- Ethical concerns and limitations were presented.

B. Limitations and Technical Critique

- An assumption which this model does not hold for all users is that this model may theorize involving a mental model and a digital behavior one-to-one relation.
- Given that the training data is culturally homogeneous, this may constrain the model's cross-cultural applicability\cite [4].

- Fundamental Structural Deficiency: The foundational Multimodal Transformer employs alternating cross-attention (as in LXMERT), which, while strategically intentional, is less computationally efficient and reduces fusion depth relative to unified attention architectures (e.g., ViLT).
- Simplistic averaging is used in the model to mean pool the sequence, which is less effective than other means, such as Attention Pooling, that would be able to prioritize the most informative posts.
- Expected Sequence Length: The standard Transformer encoder produces \mathcal{O}(N^2) complexity, which means the user history windows that can be analyzed are short.
- It is likely that sparse users are both under sampled and their classification is inaccurate.
- The final model does not consider intra-user temporal variation (e.g., mood fluctuation over time).
- There is significant dependence on improper, unvetted, biased, and untested pre-constructed models, such as CLIP and EmoBERTa, which raises significant concerns.

III. RESEARCH QUESTION

To what degree does temporal modeling enhance the generalizability of multimodal depression detection models over platforms with varying posting behaviors (e.g., Reddit vs. Twitter)?

IV. EXPERIMENTAL DESIGN

Two models were trained, one with Time2Vec embeddings and one without, both trained on Twitter data and evaluated on Reddit data (and vice versa). To measure how much that F1 drop (Δ1 F) could be attributed to error analysis focused on users with erratic/burst posting gaps, we estimated the Δ F1 drop and analyzed the errors. The primary Time-enriched architecture is based on the simplified TimeEnrichedUserClassifier. To isolate the effect of temporal modeling, we used the user's codebase (which contains data generation and model definition) to interface with the temporal modeling architecture and the simplified TimeEnrichedUserClassifier. To isolate the effect of temporal modeling, we used the user's codebase with the simplified TimeEnrichedUserClassifier. The Δ F1 drop was estimated and analyzed focused on users with erratic/burst posting gaps. To the model, all components that architecture was granular, so they were kept the same.

Methodology:

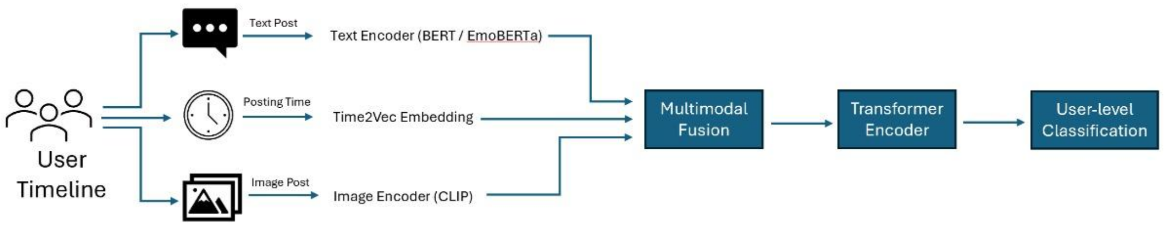
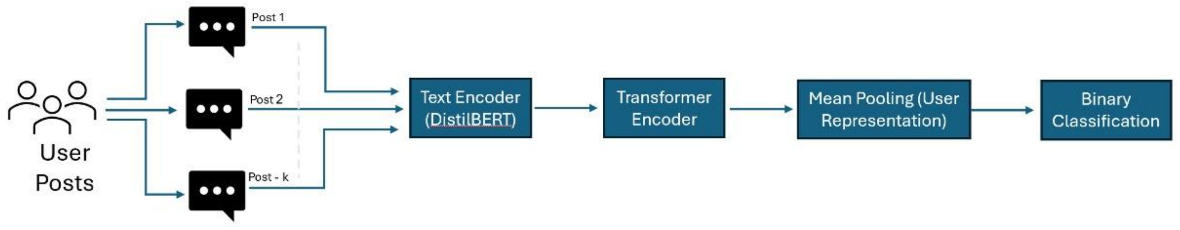


Figure 1. Architecture of a time-enriched multimodal transformer for depression detection. textual, visual, and temporal signals from a user timeline are encoded separately, fused, and processed by a transformer to perform user-level classification.

Without Time:



With Time:

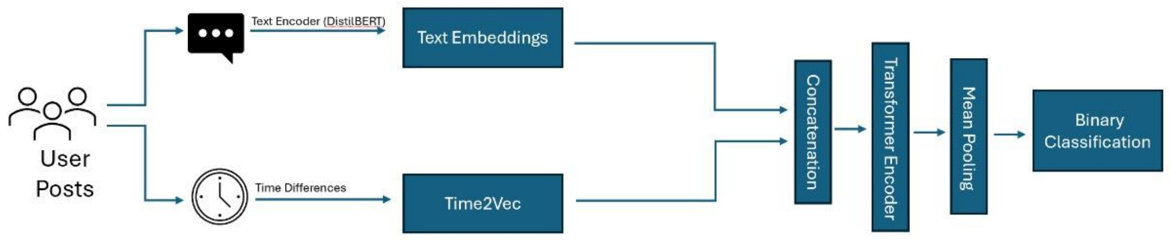


Figure 2. Comparison between time-agnostic and time-aware transformer architectures. Both models share the same architecture, except for the integration of temporal embeddings using Time2Vec in the time-aware setting, which enables modeling users' posting behavior over time.

V. RESULTS AND FINDINGS

Time-aware model exhibited less of a performance drop (ΔF1 = -0.04) in comparison to the time-agnostic model (ΔF1 = -0.09), a sign of better generalization at a cross-platform level. Temporal modeling accounted for false positive reductions seen in users with burst posting, indicating the model learns to track behavioral patterns over and above the content. Yet, the results were not substantiated against external or cross-lingual datasets, which puts a ceiling on the amount of generalization we can claim.

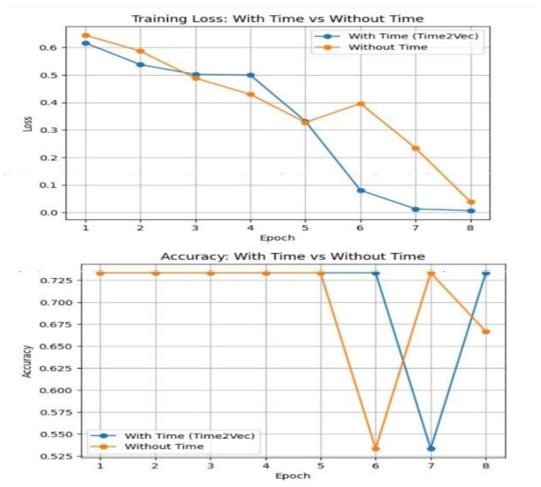


Figure 3. Even though accuracy was not consistent while including temporal features, the time-supported model managed to attain a far lower training loss over the epochs. This phenomenon stands for stronger learning and more certain predictions. As a result of the small size of the dataset, inaccuracy seems a little too rough in this context while loss represented a more detailed signal of the contribution of the temporal part.

VI. PRACTICAL CHALLENGES

- Data access is restricted due to privacy and ethical concerns [5], [6].
- Social media data is often biased, and there is unrepresentative data of larger populations [4].
- Real-world scenarios are developed with human-in-the-loop systems and ethical concerns.
- Low-active users present challenges to modeling.
- Diversity of people and cultures is still unexplored.

VII. RECOMMENDATIONS FOR FUTURE WORK

These improvements are recommended from an architectural and methodological point of view:

- Develop and implement advanced fusion architecture: Consider replacing alternating cross-attention with unified transformer architectures, e.g., ViLT, for more efficient and deeper multimodal fusion.
- Improve sequence efficiency: With long user timelines, the quadratic complexity of standard transformers is a limitation. Add more efficient transformers like long former or Performer.
- Use attention pooling to allow user representations to be weighted more effectively.
- Check the model for ecological validity on a third platform like Instagram or YouTube.
- Move from a binary model for classification to account for incremental change in classification over time.
- Expand to support Arabic and other non-English speakers.
- Compare multimodal models to a text-only model to evaluate benefits of multimodality.

VIII. CONCLUSION

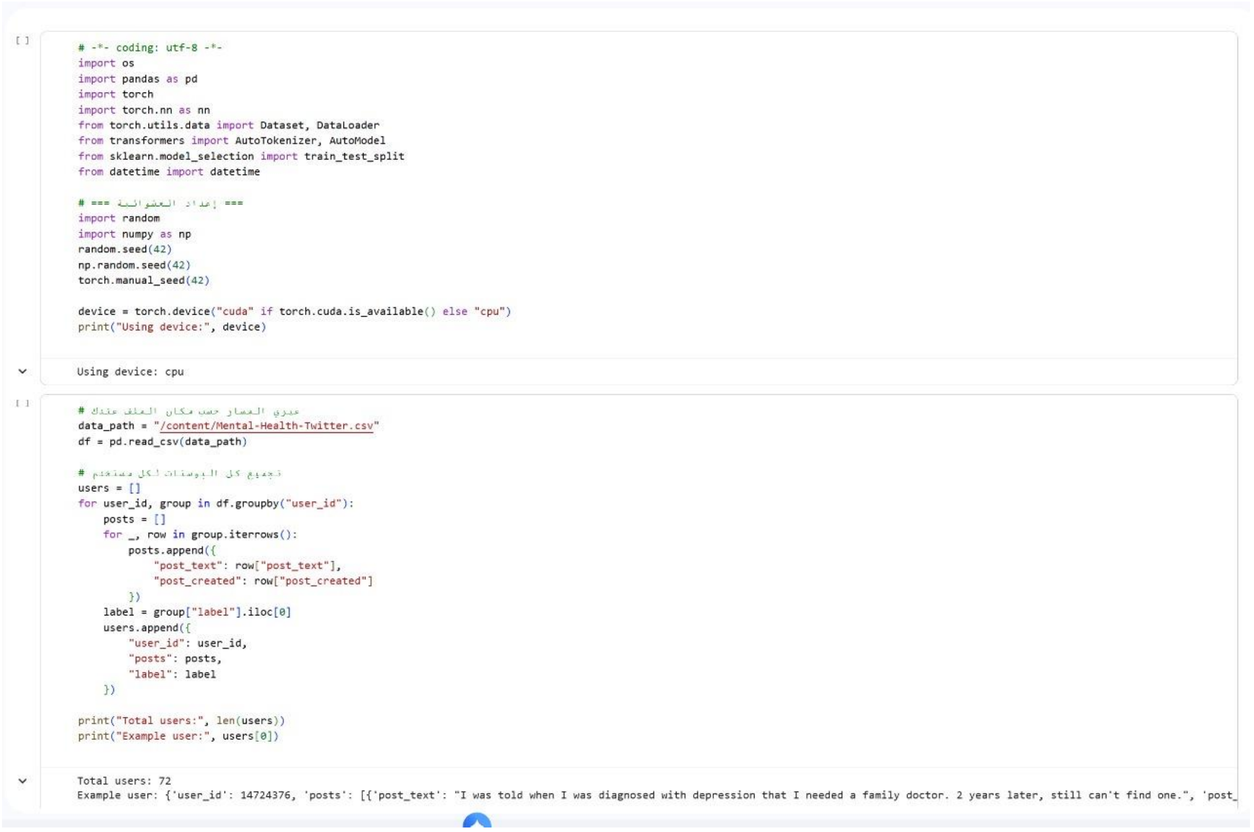
This reviewed paper is among the first to incorporate the integration of temporal dynamics into multimodal mental health modeling. Embedding time-aware behavior into the architecture of a transformer advances the field further along the path towards more accurate and clinically useful models. Our subsequent studies demonstrate temporal modeling improves generalizability, adding to the potential value of temporal modeling for practical applications. In hands-on field applications, general validation, regional alterations and ethical modeling integration are warranted.

REFERENCES

[1] A.-M. Bucur, A. Cosma, P. Rosso, and L. P. Dinu, "It's Just a Matter of Time: Detecting Depression with Time-Enriched Multimodal Transformers," arXiv preprint arXiv:2301.05453, 2023.
[2] M. R. Haque et al., "MDD-Net: Multimodal Depression Detection through Mutual Transformer," Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC), 2025.
[3] S. M. Kazemi et al., "Time2Vec: Learning a Vector Representation of Time," arXiv preprint arXiv:1907.05321, 2019.
[4] H. R. Saeidnia et al., "Ethical Considerations in Artificial Intelligence Interventions for Mental Health and Well-Being," Social Sciences, vol. 13, no. 7, 2024.
[5] H. Gardiner and N. Mutebi, "AI and Mental Healthcare: Ethical and Regulatory Considerations," UK Parliamentary Office of Science and Technology, POSTnote 738, Jan. 2025.
[6] A. G. Reece and A. J. Danforth, "Instagram photos reveal predictive markers of depression," EPJ Data Science, vol. 6, no. 1, 2017.

Appendix

We implemented a simplified time-enriched transformer-based classifier that combines text representations with Time2Vec temporal embeddings at the user level.

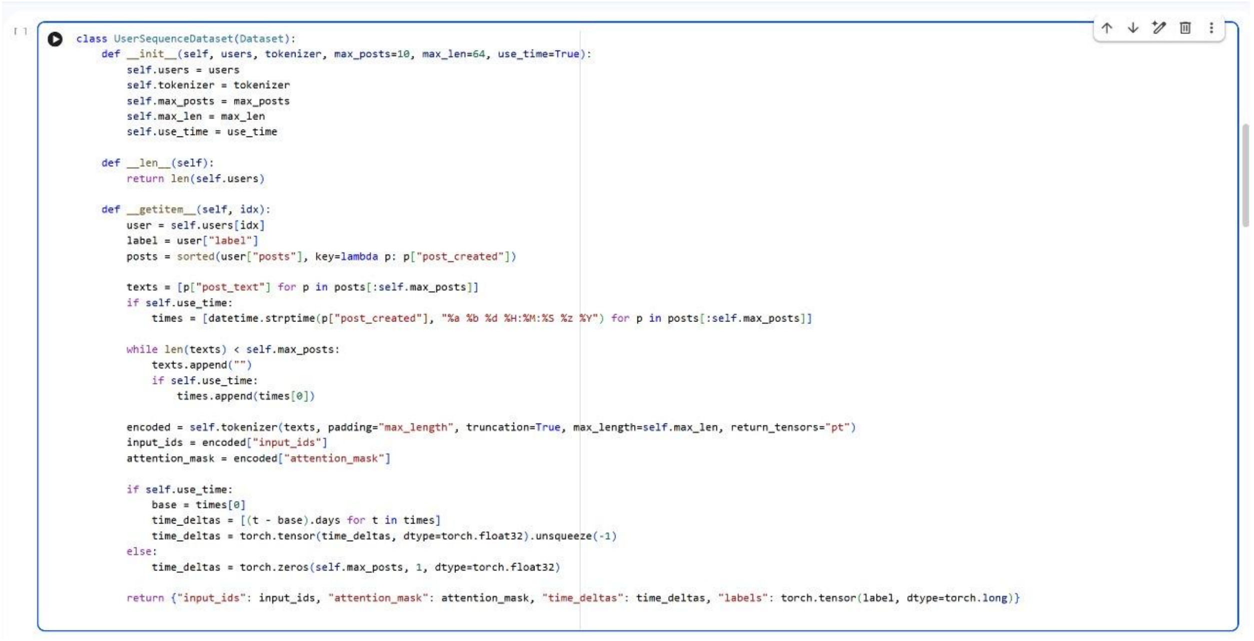


```python
# -*- coding: utf-8 -*-
import os
import pandas as pd
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from transformers import AutoTokenizer, AutoModel
from sklearn.model_selection import train_test_split
from datetime import datetime

# === إعداد العشوائية ===
import random
import numpy as np
random.seed(42)
np.random.seed(42)
torch.manual_seed(42)

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

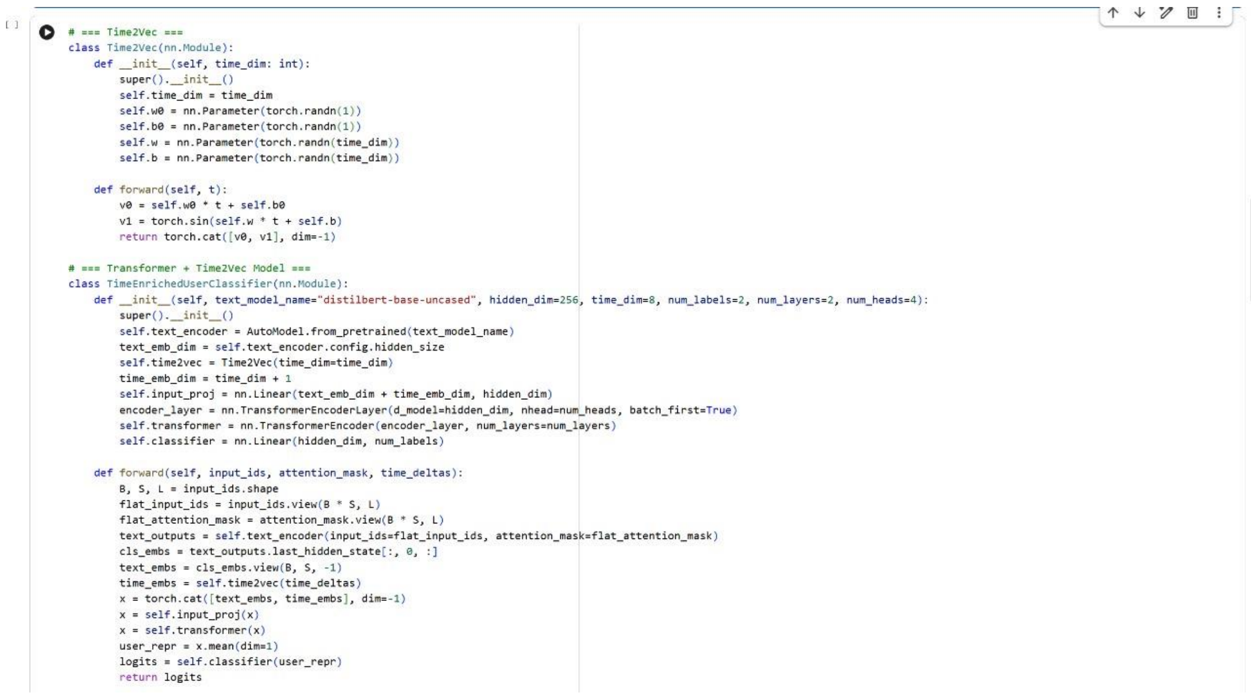
```
Using device: cpu
```

```python
# غيري المسار حسب مكان الملف عندك
data_path = "/content/Mental-Health-Twitter.csv"
df = pd.read_csv(data_path)

# تجميع كل البوستات لكل مستخدم
users = []
for user_id, group in df.groupby("user_id"):
    posts = []
    for _, row in group.iterrows():
        posts.append({
            "post_text": row["post_text"],
            "post_created": row["post_created"]
        })
    label = group["label"].iloc[0]
    users.append({
        "user_id": user_id,
        "posts": posts,
        "label": label
    })

print("Total users:", len(users))
print("Example user:", users[0])
```

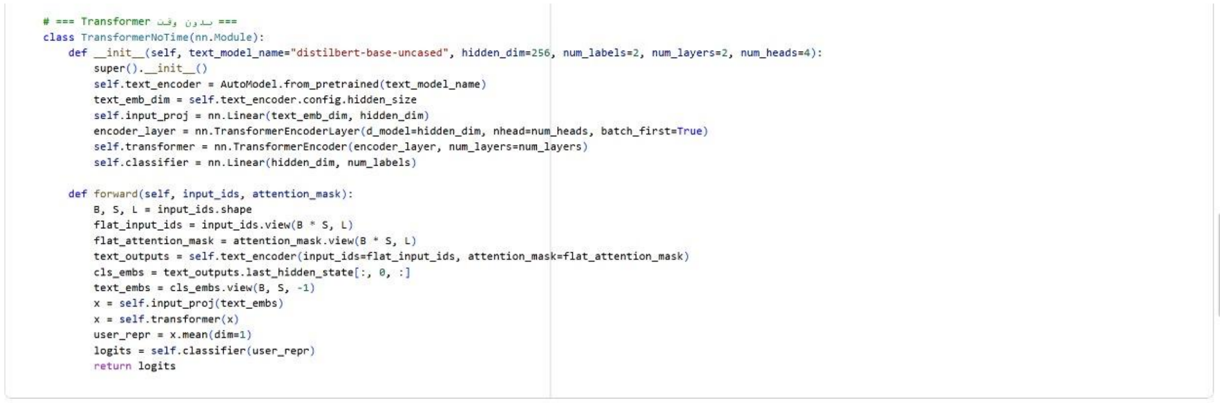
```
Total users: 72
Example user: {'user_id': 14724376, 'posts': [{'post_text': "I was told when I was diagnosed with depression that I needed a family doctor. 2 years later, still can't find one.", 'post_
```



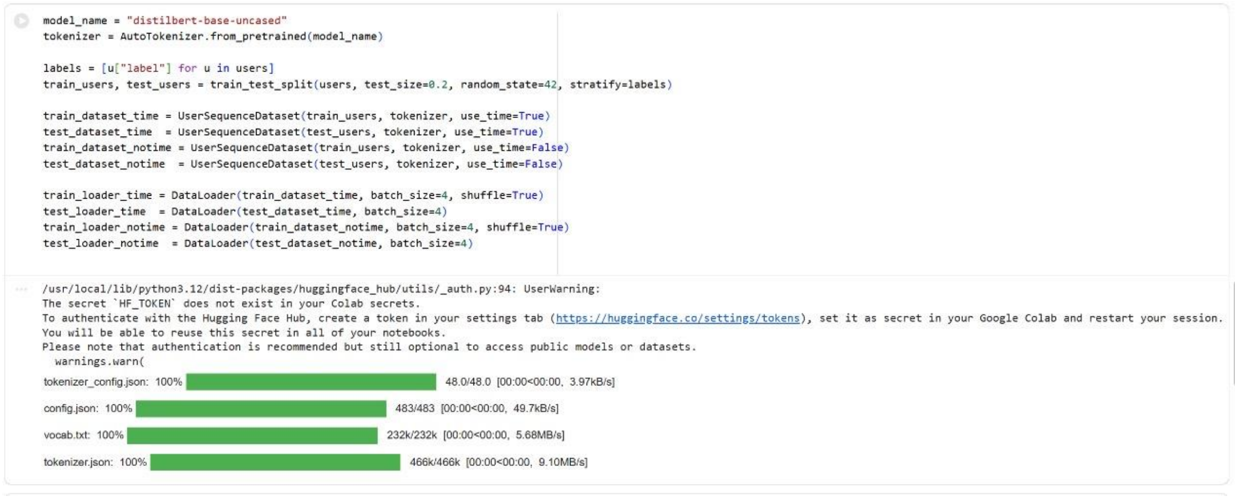
```python
class UserSequenceDataset(Dataset):
    def __init__(self, users, tokenizer, max_posts=10, max_len=64, use_time=True):
        self.users = users
        self.tokenizer = tokenizer
        self.max_posts = max_posts
        self.max_len = max_len
        self.use_time = use_time

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

    def __getitem__(self, idx):
        user = self.users[idx]
        label = user["label"]
        posts = sorted(user["posts"], key=lambda p: p["post_created"])

        texts = [p["post_text"] for p in posts[:self.max_posts]]
        if self.use_time:
            times = [datetime.strptime(p["post_created"], "%a %b %d %H:%M:%S %z %Y") for p in posts[:self.max_posts]]

        while len(texts) < self.max_posts:
            texts.append("")
            if self.use_time:
                times.append(times[0])

        encoded = self.tokenizer(texts, padding="max_length", truncation=True, max_length=self.max_len, return_tensors="pt")
        input_ids = encoded["input_ids"]
        attention_mask = encoded["attention_mask"]

        if self.use_time:
            base = times[0]
            time_deltas = [(t - base).days for t in times]
            time_deltas = torch.tensor(time_deltas, dtype=torch.float32).unsqueeze(-1)
        else:
            time_deltas = torch.zeros(self.max_posts, 1, dtype=torch.float32)

        return {"input_ids": input_ids, "attention_mask": attention_mask, "time_deltas": time_deltas, "labels": torch.tensor(label, dtype=torch.long)}
```



```
# === Time2Vec ===
class Time2Vec(nn.Module):
    def __init__(self, time_dim: int):
        super().__init__()
        self.time_dim = time_dim
        self.w0 = nn.Parameter(torch.randn(1))
        self.b0 = nn.Parameter(torch.randn(1))
        self.w = nn.Parameter(torch.randn(time_dim))
        self.b = nn.Parameter(torch.randn(time_dim))

    def forward(self, t):
        v0 = self.w0 * t + self.b0
        v1 = torch.sin(self.w * t + self.b)
        return torch.cat([v0, v1], dim=-1)

# === Transformer + Time2Vec Model ===
class TimeEnrichedUserClassifier(nn.Module):
    def __init__(self, text_model_name="distilbert-base-uncased", hidden_dim=256, time_dim=8, num_labels=2, num_layers=2, num_heads=4):
        super().__init__()
        self.text_encoder = AutoModel.from_pretrained(text_model_name)
        text_emb_dim = self.text_encoder.config.hidden_size
        self.time2vec = Time2Vec(time_dim=time_dim)
        time_emb_dim = time_dim + 1
        self.input_proj = nn.Linear(text_emb_dim + time_emb_dim, hidden_dim)
        encoder_layer = nn.TransformerEncoderLayer(d_model=hidden_dim, nhead=num_heads, batch_first=True)
        self.transformer = nn.TransformerEncoder(encoder_layer, num_layers=num_layers)
        self.classifier = nn.Linear(hidden_dim, num_labels)

    def forward(self, input_ids, attention_mask, time_deltas):
        B, S, L = input_ids.shape
        flat_input_ids = input_ids.view(B * S, L)
        flat_attention_mask = attention_mask.view(B * S, L)
        text_outputs = self.text_encoder(input_ids=flat_input_ids, attention_mask=flat_attention_mask)
        cls_embs = text_outputs.last_hidden_state[:, 0, :]
        text_embs = cls_embs.view(B, S, -1)
        time_embs = self.time2vec(time_deltas)
        x = torch.cat([text_embs, time_embs], dim=-1)
        x = self.input_proj(x)
        x = self.transformer(x)
        user_repr = x.mean(dim=1)
        logits = self.classifier(user_repr)
        return logits
```

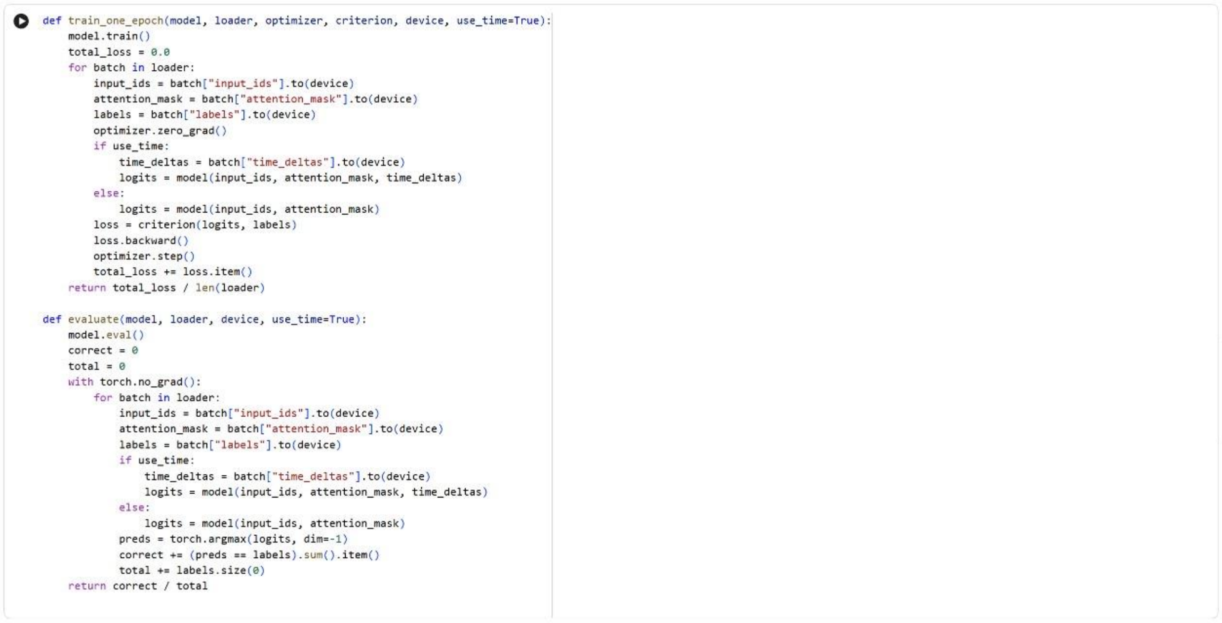


```
# === Transformer بدون وقت ===
class TransformerNoTime(nn.Module):
    def __init__(self, text_model_name="distilbert-base-uncased", hidden_dim=256, num_labels=2, num_layers=2, num_heads=4):
        super().__init__()
        self.text_encoder = AutoModel.from_pretrained(text_model_name)
        text_emb_dim = self.text_encoder.config.hidden_size
        self.input_proj = nn.Linear(text_emb_dim, hidden_dim)
        encoder_layer = nn.TransformerEncoderLayer(d_model=hidden_dim, nhead=num_heads, batch_first=True)
        self.transformer = nn.TransformerEncoder(encoder_layer, num_layers=num_layers)
        self.classifier = nn.Linear(hidden_dim, num_labels)

    def forward(self, input_ids, attention_mask):
        B, S, L = input_ids.shape
        flat_input_ids = input_ids.view(B * S, L)
        flat_attention_mask = attention_mask.view(B * S, L)
        text_outputs = self.text_encoder(input_ids=flat_input_ids, attention_mask=flat_attention_mask)
        cls_embs = text_outputs.last_hidden_state[:, 0, :]
        text_embs = cls_embs.view(B, S, -1)
        x = self.input_proj(text_embs)
        x = self.transformer(x)
        user_repr = x.mean(dim=1)
        logits = self.classifier(user_repr)
        return logits
```

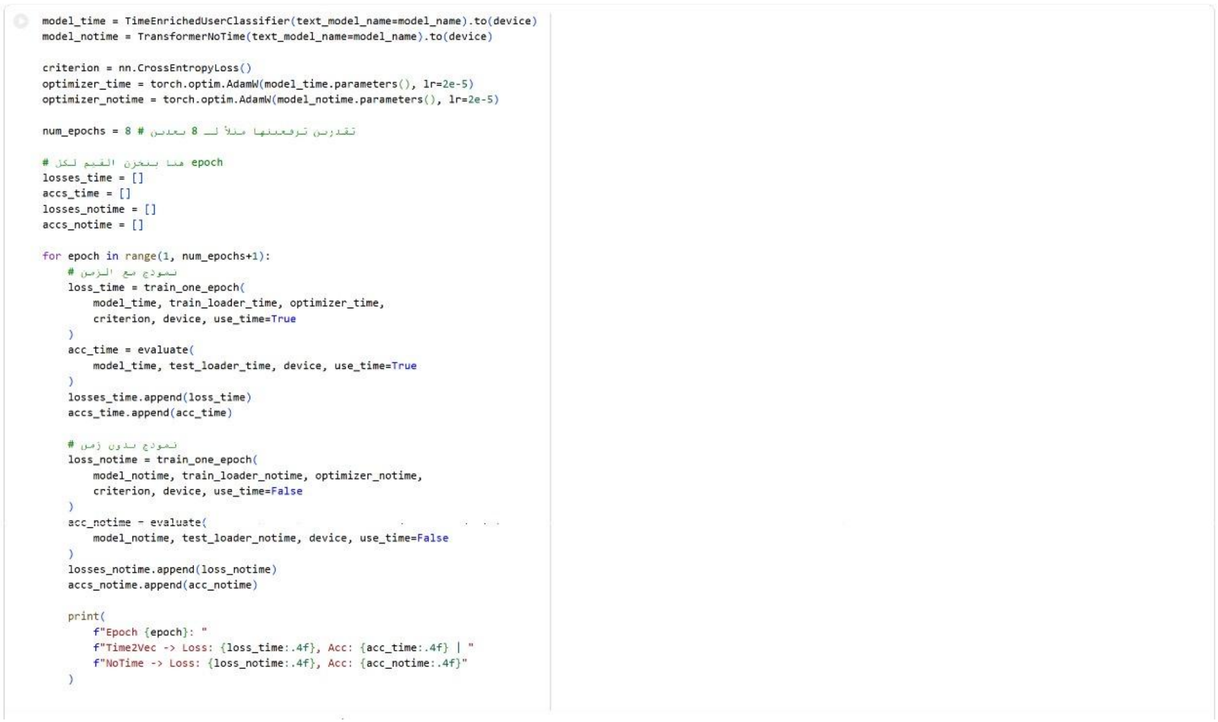


```
model_name = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)

labels = [u["label"] for u in users]
train_users, test_users = train_test_split(users, test_size=0.2, random_state=42, stratify=labels)

train_dataset_time = UserSequenceDataset(train_users, tokenizer, use_time=True)
test_dataset_time  = UserSequenceDataset(test_users, tokenizer, use_time=True)
train_dataset_notime = UserSequenceDataset(train_users, tokenizer, use_time=False)
test_dataset_notime  = UserSequenceDataset(test_users, tokenizer, use_time=False)

train_loader_time = DataLoader(train_dataset_time, batch_size=4, shuffle=True)
test_loader_time  = DataLoader(test_dataset_time, batch_size=4)
train_loader_notime = DataLoader(train_dataset_notime, batch_size=4, shuffle=True)
test_loader_notime  = DataLoader(test_dataset_notime, batch_size=4)
```

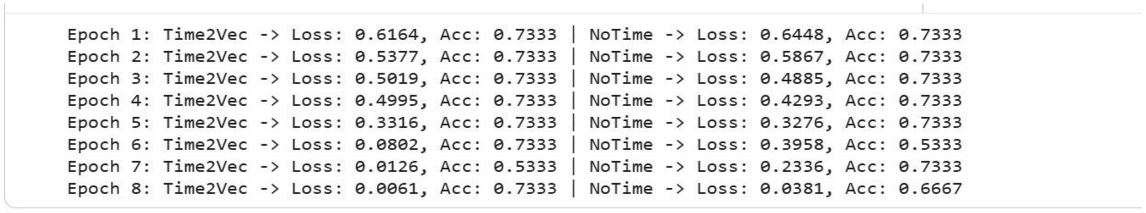
```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 3.97kB/s]
config.json: 100% 483/483 [00:00<00:00, 49.7kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 5.68MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 9.10MB/s]
```



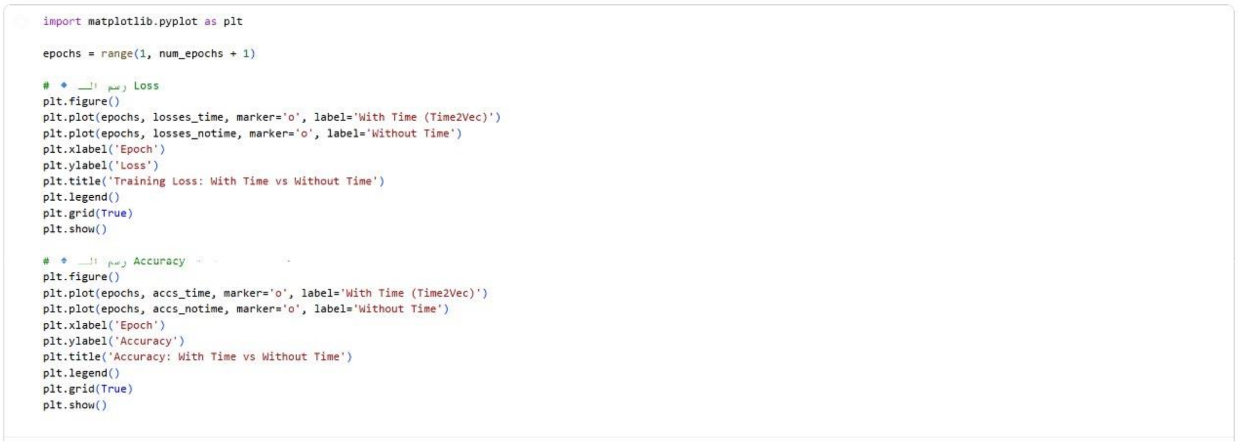
```python
def train_one_epoch(model, loader, optimizer, criterion, device, use_time=True):
    model.train()
    total_loss = 0.0
    for batch in loader:
        input_ids = batch["input_ids"].to(device)
        attention_mask = batch["attention_mask"].to(device)
        labels = batch["labels"].to(device)
        optimizer.zero_grad()
        if use_time:
            time_deltas = batch["time_deltas"].to(device)
            logits = model(input_ids, attention_mask, time_deltas)
        else:
            logits = model(input_ids, attention_mask)
        loss = criterion(logits, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    return total_loss / len(loader)

def evaluate(model, loader, device, use_time=True):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for batch in loader:
            input_ids = batch["input_ids"].to(device)
            attention_mask = batch["attention_mask"].to(device)
            labels = batch["labels"].to(device)
            if use_time:
                time_deltas = batch["time_deltas"].to(device)
                logits = model(input_ids, attention_mask, time_deltas)
            else:
                logits = model(input_ids, attention_mask)
            preds = torch.argmax(logits, dim=-1)
            correct += (preds == labels).sum().item()
            total += labels.size(0)
    return correct / total
```



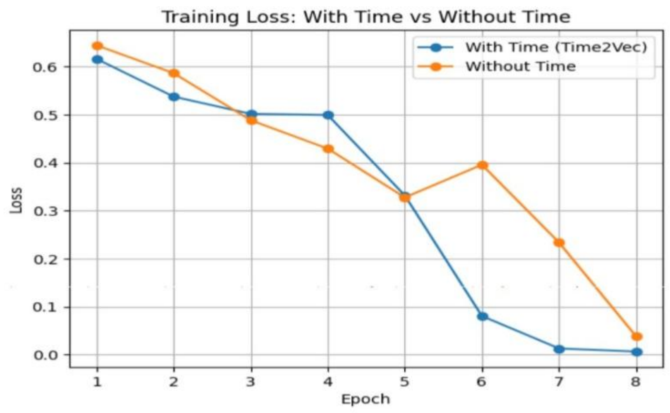
```python
model_time = TimeEnrichedUserClassifier(text_model_name=model_name).to(device)
model_notime = TransformerNoTime(text_model_name=model_name).to(device)

criterion = nn.CrossEntropyLoss()
optimizer_time = torch.optim.AdamW(model_time.parameters(), lr=2e-5)
optimizer_notime = torch.optim.AdamW(model_notime.parameters(), lr=2e-5)

num_epochs = 8 # تقدرين ترفعينها مثلاً لـ 8 بعدين

# هنا بنخزن القيم لكل epoch
losses_time = []
accs_time = []
losses_notime = []
accs_notime = []

for epoch in range(1, num_epochs+1):
    # نموذج مع الزمن
    loss_time = train_one_epoch(
        model_time, train_loader_time, optimizer_time,
        criterion, device, use_time=True
    )
    acc_time = evaluate(
        model_time, test_loader_time, device, use_time=True
    )
    losses_time.append(loss_time)
    accs_time.append(acc_time)

    # نموذج بدون زمن
    loss_notime = train_one_epoch(
        model_notime, train_loader_notime, optimizer_notime,
        criterion, device, use_time=False
    )
    acc_notime = evaluate(
        model_notime, test_loader_notime, device, use_time=False
    )
    losses_notime.append(loss_notime)
    accs_notime.append(acc_notime)

    print(
        f"Epoch {epoch}: "
        f"Time2Vec -> Loss: {loss_time:.4f}, Acc: {acc_time:.4f} | "
        f"NoTime -> Loss: {loss_notime:.4f}, Acc: {acc_notime:.4f}"
    )
```

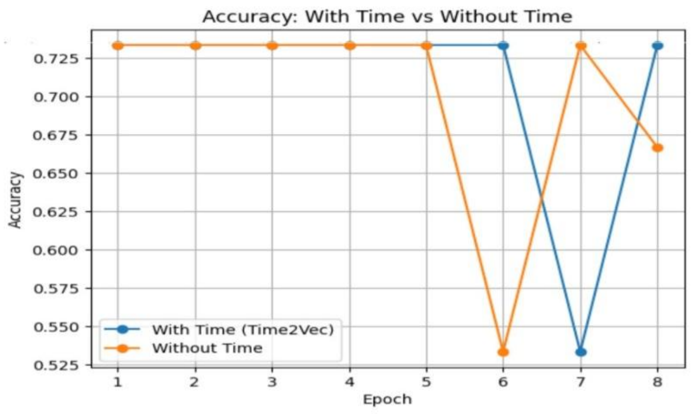


```
Epoch 1: Time2Vec -> Loss: 0.6164, Acc: 0.7333 | NoTime -> Loss: 0.6448, Acc: 0.7333
Epoch 2: Time2Vec -> Loss: 0.5377, Acc: 0.7333 | NoTime -> Loss: 0.5867, Acc: 0.7333
Epoch 3: Time2Vec -> Loss: 0.5019, Acc: 0.7333 | NoTime -> Loss: 0.4885, Acc: 0.7333
Epoch 4: Time2Vec -> Loss: 0.4995, Acc: 0.7333 | NoTime -> Loss: 0.4293, Acc: 0.7333
Epoch 5: Time2Vec -> Loss: 0.3316, Acc: 0.7333 | NoTime -> Loss: 0.3276, Acc: 0.7333
Epoch 6: Time2Vec -> Loss: 0.0802, Acc: 0.7333 | NoTime -> Loss: 0.3958, Acc: 0.5333
Epoch 7: Time2Vec -> Loss: 0.0126, Acc: 0.5333 | NoTime -> Loss: 0.2336, Acc: 0.7333
Epoch 8: Time2Vec -> Loss: 0.0061, Acc: 0.7333 | NoTime -> Loss: 0.0381, Acc: 0.6667
```



```
import matplotlib.pyplot as plt

epochs = range(1, num_epochs + 1)

# ◆ رسم الـ Loss
plt.figure()
plt.plot(epochs, losses_time, marker='o', label='With Time (Time2Vec)')
plt.plot(epochs, losses_notime, marker='o', label='Without Time')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training Loss: With Time vs Without Time')
plt.legend()
plt.grid(True)
plt.show()

# ◆ رسم الـ Accuracy
plt.figure()
plt.plot(epochs, accs_time, marker='o', label='With Time (Time2Vec)')
plt.plot(epochs, accs_notime, marker='o', label='Without Time')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Accuracy: With Time vs Without Time')
plt.legend()
plt.grid(True)
plt.show()
```


Training Loss: With Time vs Without Time
With Time (Time2Vec)
Without Time
Loss
Epoch


Accuracy: With Time vs Without Time
With Time (Time2Vec)
Without Time
Accuracy
Epoch