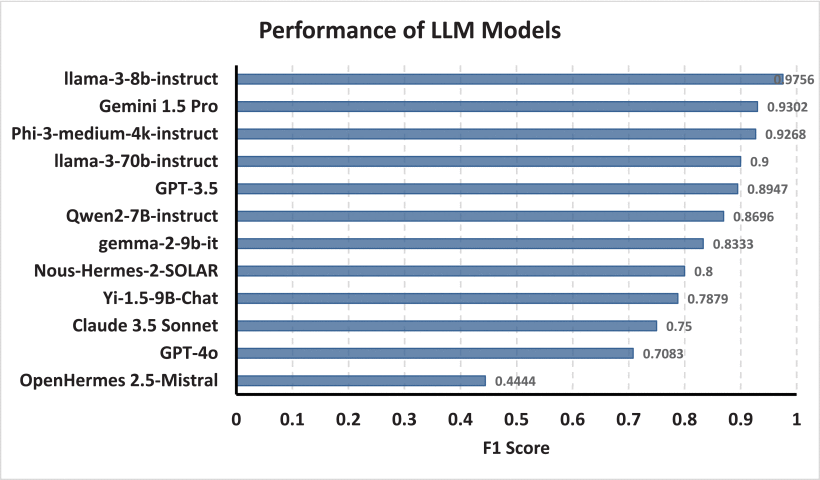
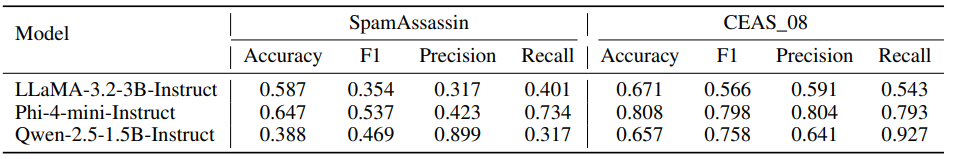
1. **Literatür Taraması**

Phishing Mail Detection hakkında 2025 yılında yayınlanmış 4 adet makale incelenmiştir. Literatür taramasında özellikle son yıllarda ML yöntemlerinden ziyade daha çok transormer tabanlı yöntemler ile ilgili çalışıldığı görülmüştür.

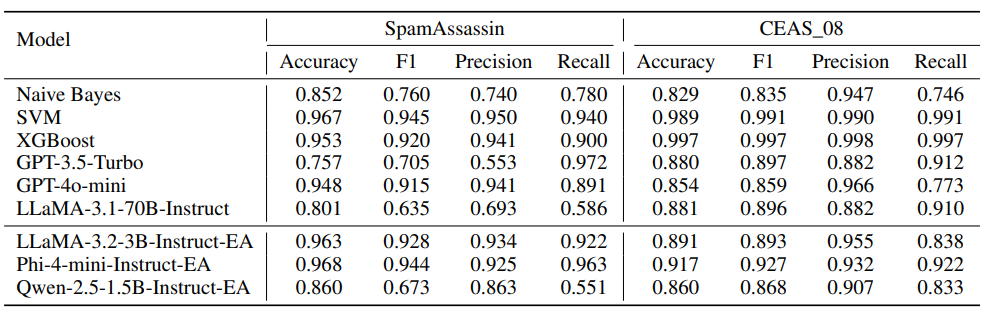
* “J. Zhang, P. Wu, J. London and D. Tenney, "Benchmarking and Evaluating Large Language Models in Phishing Detection for Small and Midsize Enterprises: A Comprehensive Analysis," in IEEE Access, vol. 13, pp. 28335-28352, 2025, doi: 10.1109/ACCESS.2025.3540075. keywords: {Phishing;Electronic mail;Data models;Large language models;Computer security;Security;Accuracy;Unsolicited e-mail;Benchmark testing;Analytical models;Large language models (LLMs);generative artificial intelligence (AI);small-to-medium enterprise;cybersecurity;technology management;social engineering;phishing detection}” makalesinde özetle;
  + Fine-tune yapmadan, temel sohbet/instruct LLM’lerini hazır promt şablonları ile doğrudan kullanıp phishing tespiti yapmak amaçlanmıştır.Sonuçlar aşağıdaki gibi üretilmiş olup en büyük modelin en iyi sonuç vermeyebileceği vurgulanmıştır.



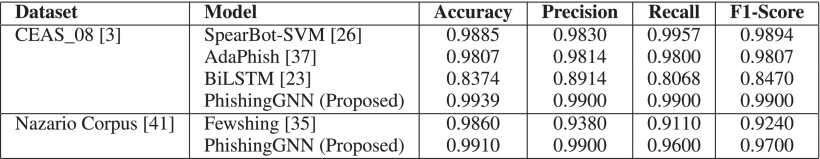
* “Zijie Lin, Zikang Liu, Hanbo Fan, IMPROVING PHISHING EMAIL DETECTION PERFORMANCE OF SMALL LARGE LANGUAGE MODELS” makalesinde özetle;
  + Büyük ve pahalı LLM modelleri yerine küçük boyutlu LLM modelleri ile sınıflandırma yapmak hedeflenmiştir. Bunun için Promt mühendisliği, (Explanation-Augmented Fine-Tuning, EAFT) + LoRA ile her eğitim örneği “e-posta + etiket + kısa açıklama” üçlüsüne dönüştürülüp düşük maliyetli LoRA ile ince ayarlama ve çoğunluk oyu ve/veya çıktı güvenine dayalı seçimle modellerin birleştirilmesi sağlanmıştır.Sonuçta sadece promt ile özellikle zorlu veri setlerinde küçük LLM’lerin sınırlı kaldığı, EAFT + LoRA sonrasında doğruluğun ciddi oranda yükseldiği belirtilmiştir.



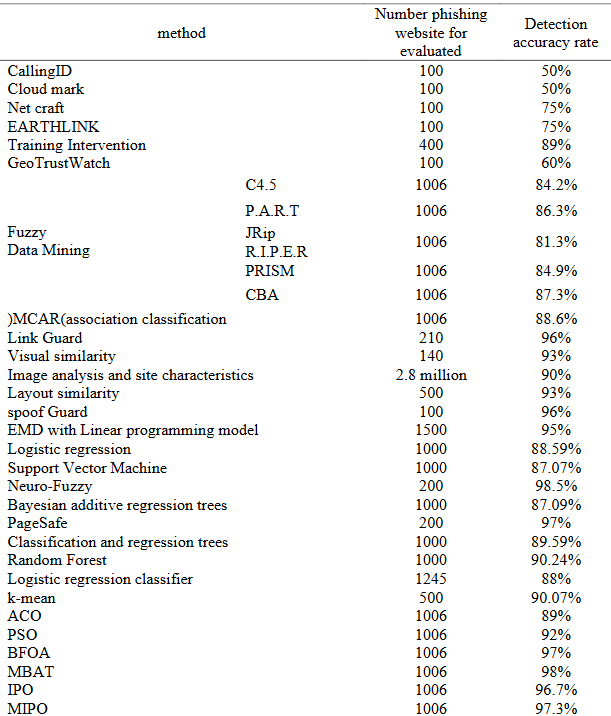
Sadece Prompt İle Sonuçlar



EAFT+LoRA Sonrası Sonuçlar

* “PhishingGNN: Phishing Email Detection Using Graph Attention Networks and Transformer-Based Feature Extraction” makalesinde özetle;
  + “PhishingGNN, e-posta metnini DistilBERT ile anlamsal uzaya gömüp, metni kelime düğümleri ve ardışıklık kenarlarıyla grafa çevirerek GAT ile sınıflandıran hibrit bir mimari sunmaktadır.

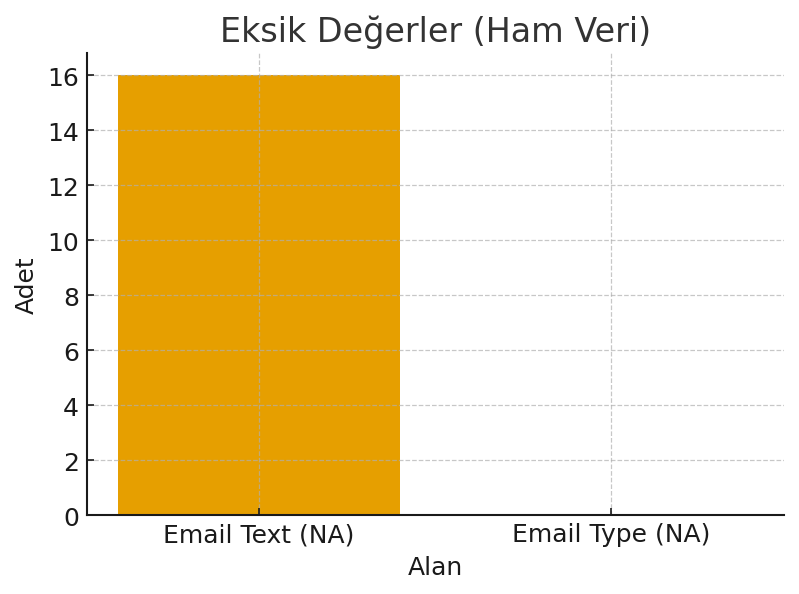
Sonuçlar

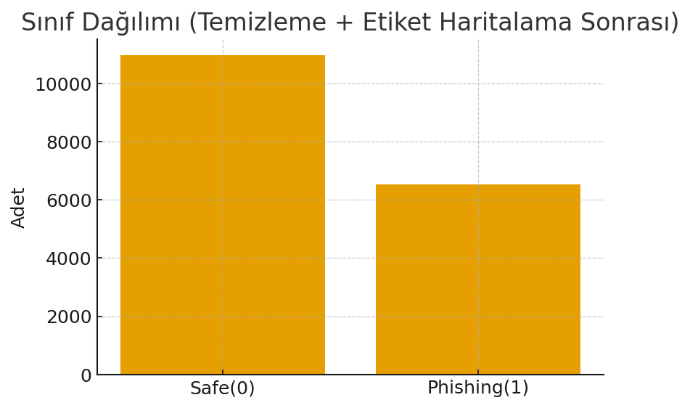
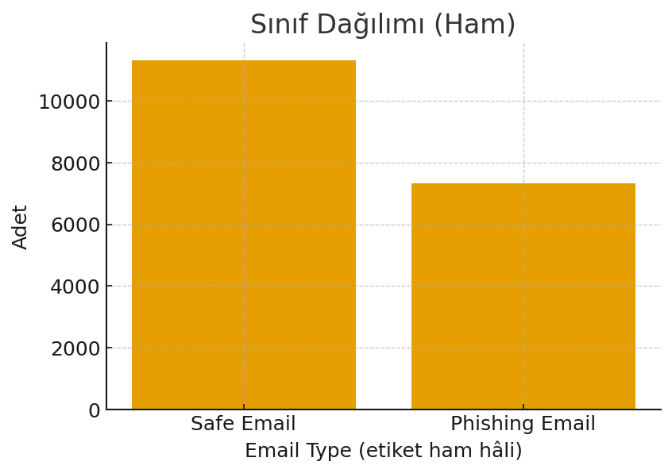
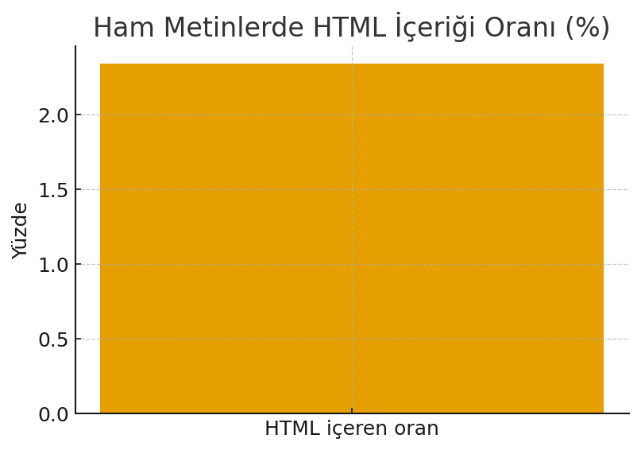
* “Majid Abdolrazzagh-Nezhad, Nafise Langarib, Phishing Detection Techniques: A review” makalesinde özetle;
  + E-postalardan gövde-başlık-URL vb. gibi hibrit özellikler çıkarılıp klasik ML (LR/SVM/RF/GBM) ile phishing tespiti yapmaya yönelik özellik mühendisliği odaklı bir sistem önerilmiştir. HTML/URL temizliği ve deduplikasyon dahil önişleme sonrası, hibrit özellikler tekil özelliklere kıyasla belirgin kazanım sağladığı belirtilmiştir. 

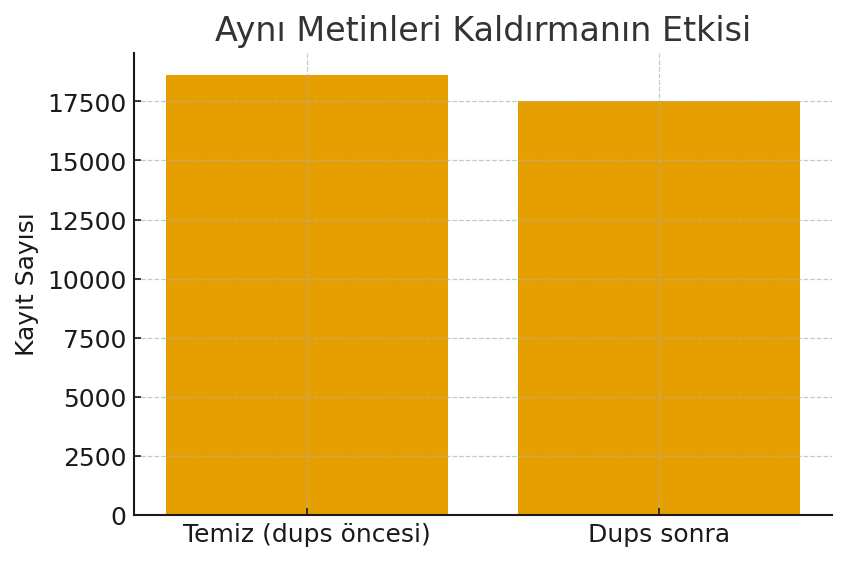
Yöntemler ve Sonuçları

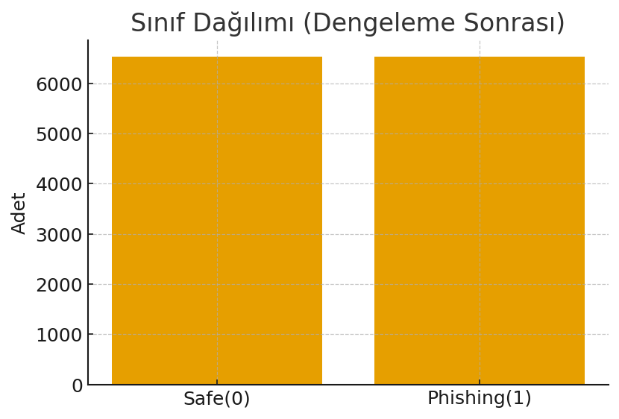
1. **Veri Ön İşleme**

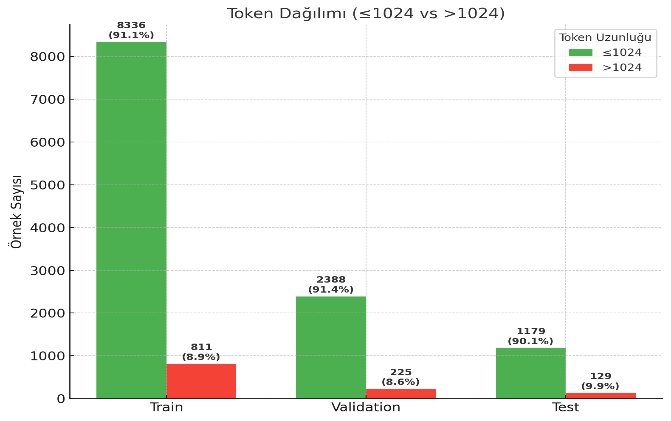
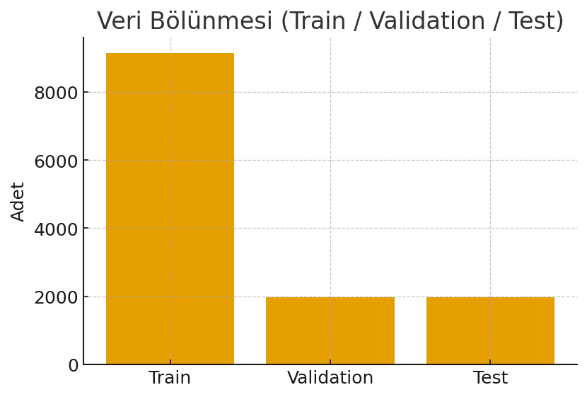
İlk olarak veri seti indirilip incelendi ve aşağıdaki durumlar tespit edildi.











Önceki sayfadaki grafiklerde görüldüğü üzere;

* Veri setinde eksik ve tekrarlı değerler olduğu,
* Bazı unicode bozulmalarının tespit edildiği,
* Sınıfların dağılımında veri ön işleme ve sonrasında kararsızlık olduğu,
* GPT-2 modeli fine-tune edileceğinden (context size 1024), tokenların veri ön işleme sonrası dağılımlarına bakıldığında eğitim verisinin ~%92’sinin token uzunluğunun 1024 ve küçük olduğu kalanın ise yüksek olduğu gözlemlenmiştir.

Bunların çözümü için “PhishingEmailPreprocessor” sınıfı yazılmış olup veri önce burada işleme sokulmuştur. Ön işleme adımlarından tokenize işlemi burada yapılmış, padding ve/veya truncate işlemi daha sonra yapılmıştır. Verinin %70’i eğitim, %20’si validation ve %10’u test olarak parçalanmıştır.

1. **Ön İşlemden Geçirilen Verinin Data Loader’a Yüklenmesi**

“EmailDataset” ve “EmailDataLoader” sınıfları bu amaç ile yazılmıştır. Burada ilk olarak 1024 tokendan küçük olanlar padding ile (50256) 1024 tokena çıkarılmış, eşit olanlar aynı bırakılmış ve büyük olanların ilk 1024 tokenı alınmıştır. Eğitim sürecinde bunun gereksiz yavaşlığa neden olduğu görülmüş ve “EmailDataLoader” sınıfına “make\_collate\_fn” fonksiyonu yazılmıştır.

Bu fonksiyon aynı batch içerisindeki verilerinden en uzun tokena sahip olanı bulup diğerlerini sadece padding işlemi yaparak bunun sayısına getirmekte olup gereksiz padding işleminden kaçınılmıştır.

Eğitimde kullanılan GPU’nun VRAM’i batch size 16’yı yapabilecek durumda olduğundan batch size burada 16 seçilmiştir.

1. **Model**

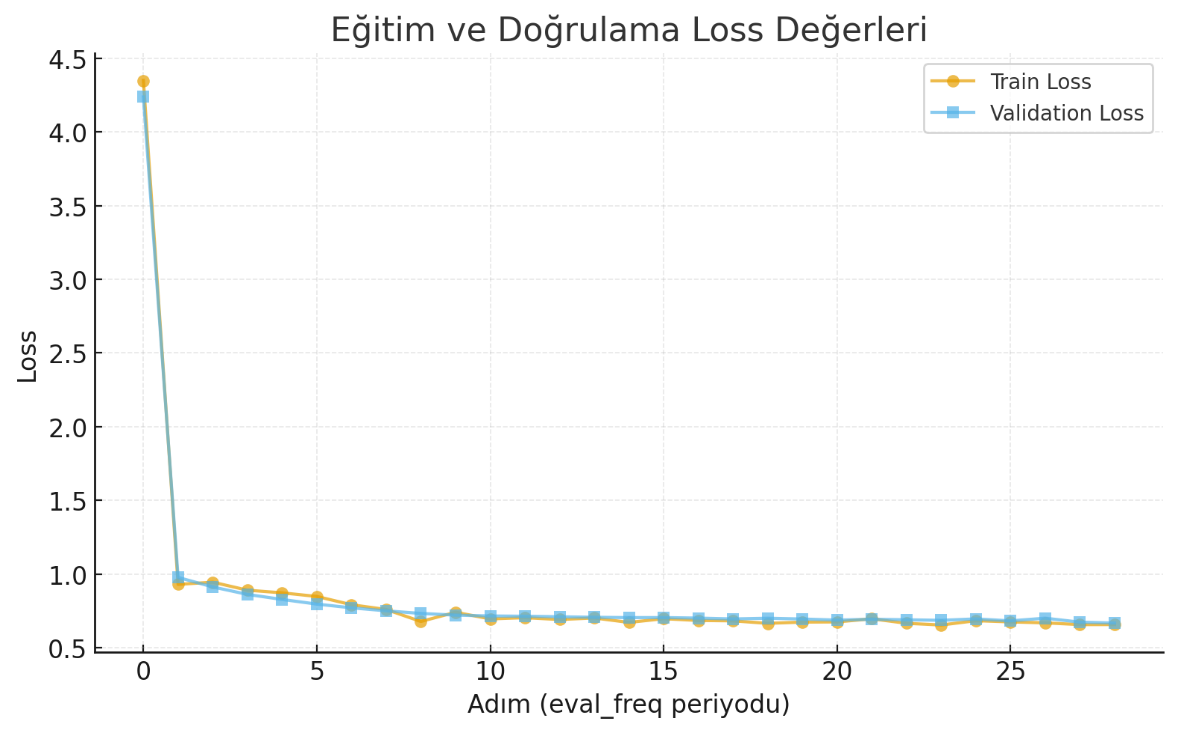
Bu kapsamda “MultiHeadAttention”, “GELU”, “FeedForward”, “LayerNorm”, “TransformerBlock”, “GPTModel” sınıfları uygulanmıştır. Model olarak gpt-2 small kullanılmış olup, “GPTModel” sınıfına config olarak gpt-2 small configleri geçilmiştir.

Eğitime başlanmadan önce tüm ağırlıklar dondurulmuş, gpt-2 small modelinin son katmanı 2 çıkışlı (phising, safe) network ile değiştirilmiştir. Son eklenen blok, öncesindeki “LayerNorm” bloğu ve son transformer bloğunun eğitilmesine karar verilmiştir. Eğitimde yeterli başarı alınamazsa 1 tane daha transformer bloğunun eğitime katılması değerlendirilebilecektir.

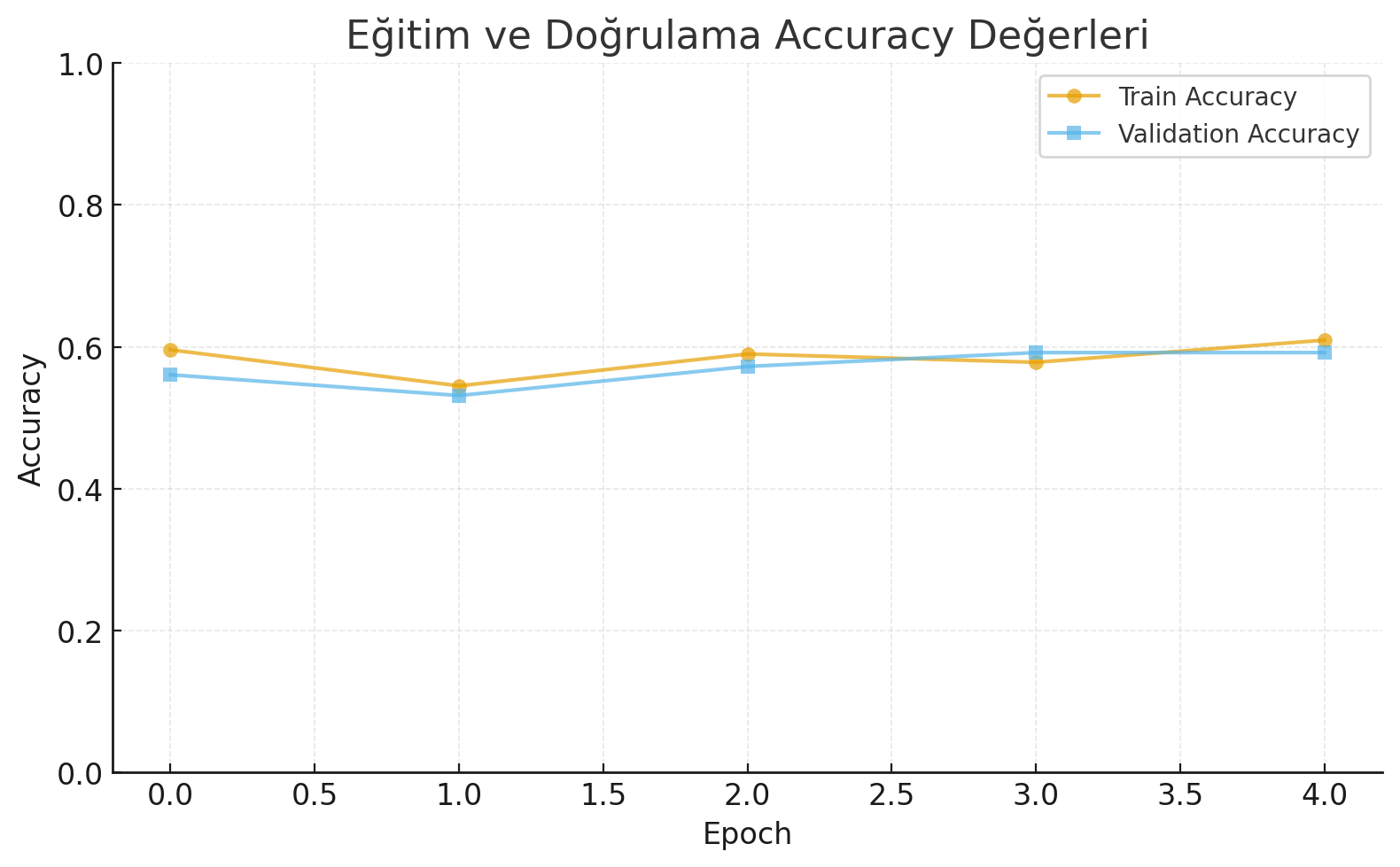
1. **Eğitim**

Tüm llm fine tune işlemleri NVIDIA GeForce RTX 3060 Ekran Kartı ve gpt-2 small modeli ile yapılmıştır. İlk olarak yapılan eğitimde (84 dakika sürmüştür);

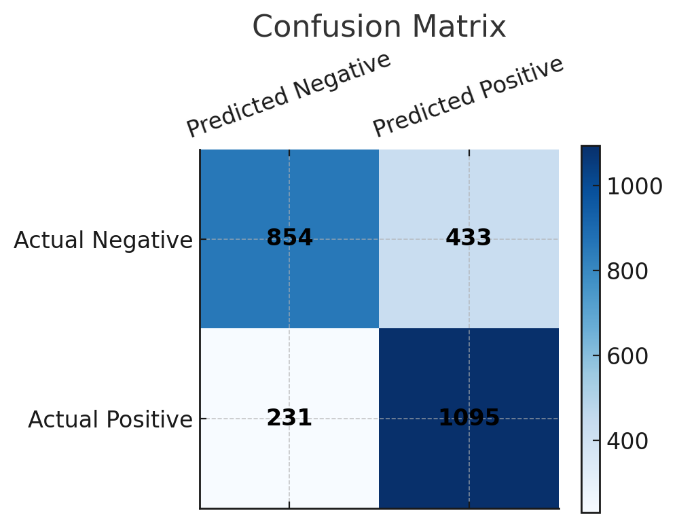
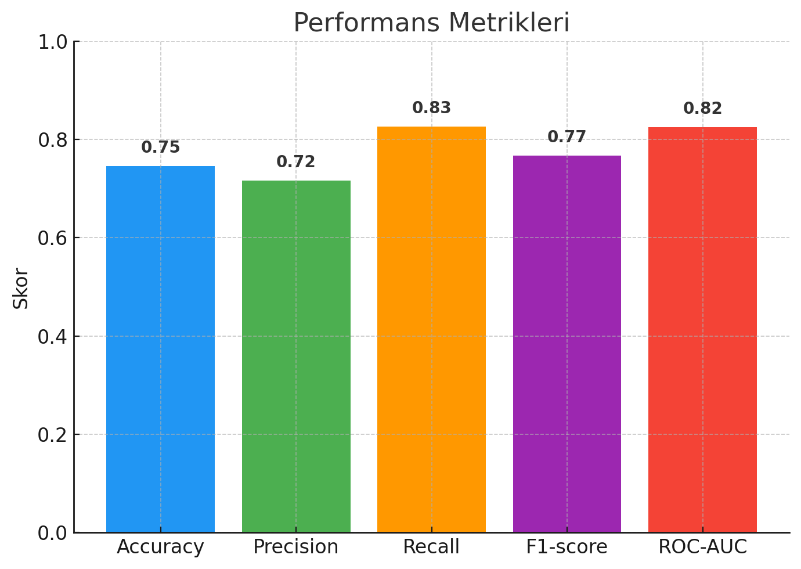
* Learning rate; 5e-5
* Weight decay; 0.1
* Num epoch; 5 seçilmiştir. Eğitim sonuçları aşağıdaki gibi olmuştur.



1.Eğitim Loss Sonuçları



1.Eğitim Accuracy Sonuçları



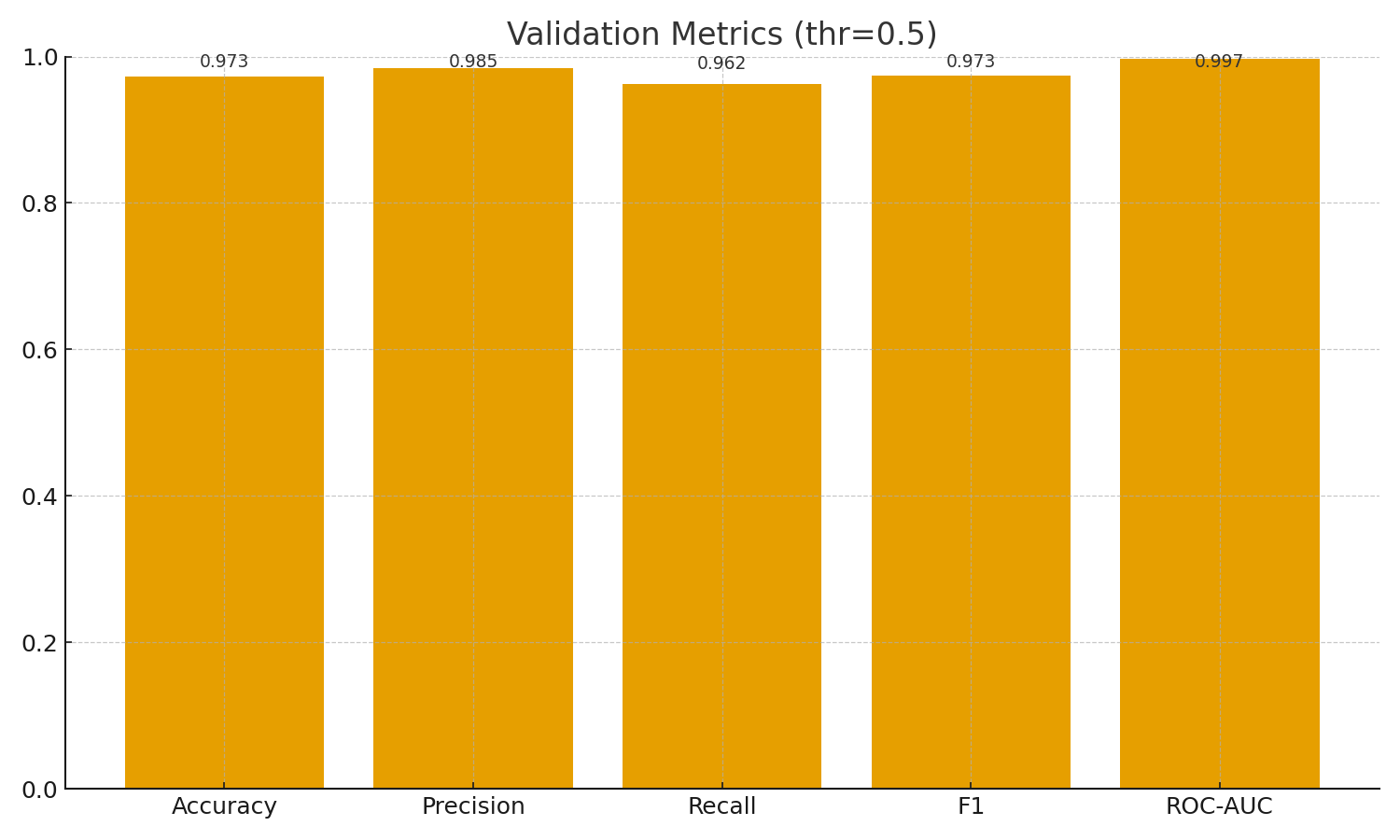
1.Eğitim Performans Metrikleri

1.Eğitim Confusion Matrix

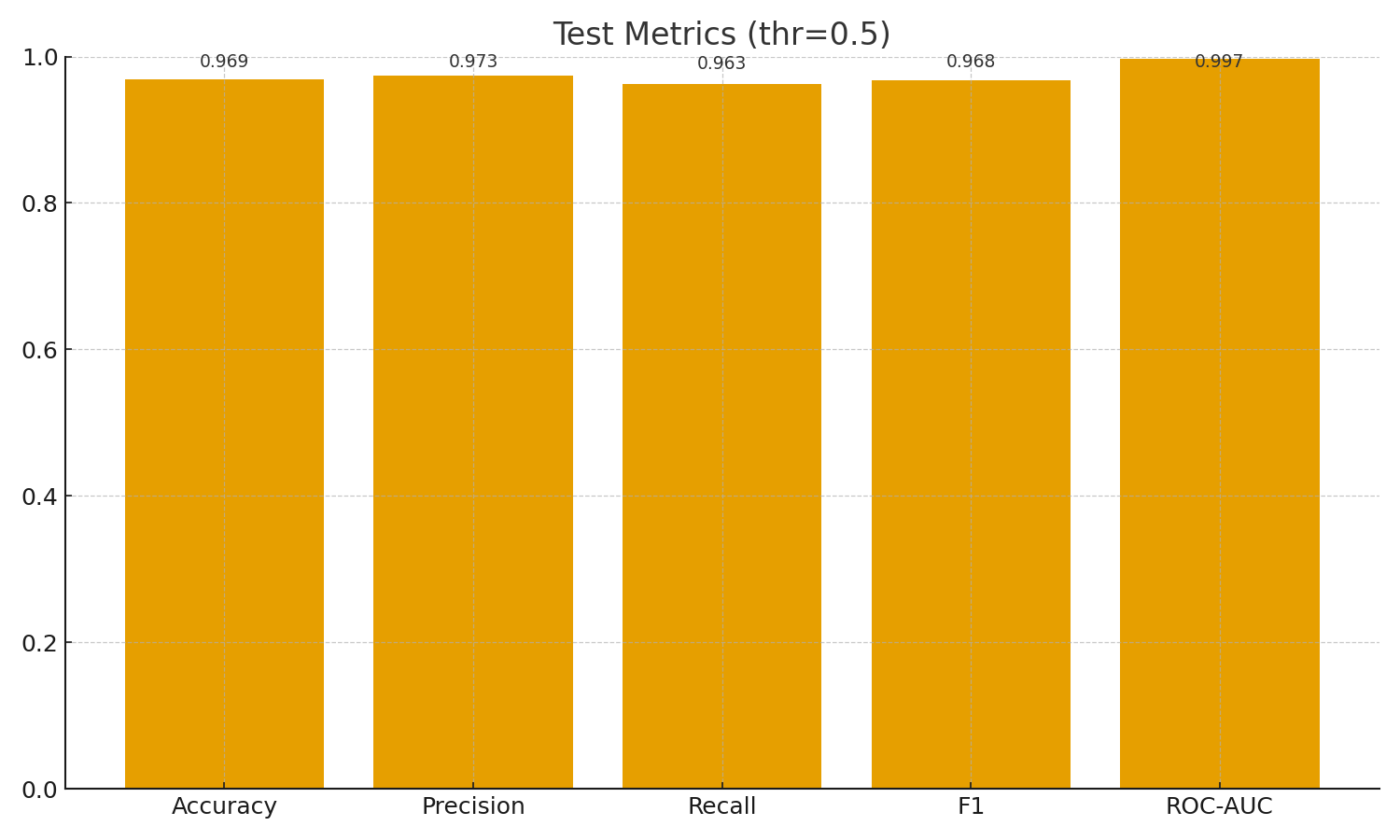
Sonuçlar neticesinde her ne kadar loss’ta düşüş yaşanmış olsa da performans metriklerinde istenilen sonuçlar alınamamıştır. Yapılan literatür araştırmaları ve yazılımda yapılan incelemeler sonucunda;

* learning rate için daha küçük bir değerle başlamasının daha uygun olacağına karar verilmiş ve 2.Eğitim için 1e-5 seçilmiştir.
* weight decay için ağırlıklarda %10 azaltmanın ilk etapta çok olduğu düşünülmüş ve 2.Eğitim için bu değer 0.01 seçilmiştir.
* Yazılımın “MultiHeadAttention” kısmında padding mask olmadığı görülmüştür. Modelin son çıktısına bakarak (logits = model(input\_batch)[:, -1, :]) karar verildiği için ve veri ön işlemede padding yapıldığından bu karar verme işlemini yaparken her örnekte PAD olmayan son token indexini bulup logitsi oradan alınmıştır. (“last\_valid\_logits” fonksiyonu)
* Yapılan araştırmalar sonucunda “LayerNorm” bloğuna genelde weight decay uygulanmaması, learning rate parametresini modelin son katmanı ile transformer bloğunun son katmanı için farklı (genelde son katmana daha büyük lr) verilmesinin fayda sağlayabileceği görülmüş olup eğitimin başarısız olma durumunda denenmeye karar verilmiştir.
* 2.Eğitimde num epoch 20 seçilmiş (ilk eğitimde overfit vb. sorunların olmadığı ve son epochlarda dahi lossun düştüğü görüldüğüden) ve early stopping ayarlanmıştır. 3 epoch boyunca validation loss 0.001 kadar düşmez ise eğitim durdurulmuştur. Her iyileşmede checkpoint yazılmıştır.
* 2.Eğitim 421 dakika sürmüştür.
* Eğitim sırasındaki aşağıdaki çıktıları incelediğimizde;
  + Ep 1 (Step 000200): Train loss 1.225, Val loss 1.574
  + Ep 1 (Step 000400): Train loss 0.461, Val loss 0.828
  + **[Epoch 1] mean Train loss 0.716, mean Val loss 0.719**
  + Training accuracy: 64.06% | Validation accuracy: 71.88%
  + Ep 2 (Step 000600): Train loss 0.747, Val loss 0.692
  + Ep 2 (Step 000800): Train loss 0.634, Val loss 0.596
  + Ep 2 (Step 001000): Train loss 0.530, Val loss 0.522
  + **[Epoch 2] mean Train loss 0.404, mean Val loss 0.416**
  + Training accuracy: 84.38% | Validation accuracy: 78.12%
  + Ep 3 (Step 001200): Train loss 0.449, Val loss 0.456
  + Ep 3 (Step 001400): Train loss 0.234, Val loss 0.402
  + Ep 3 (Step 001600): Train loss 0.162, Val loss 0.367
  + **[Epoch 3] mean Train loss 0.231, mean Val loss 0.251**
  + Training accuracy: 93.75% | Validation accuracy: 79.69%
  + Ep 4 (Step 001800): Train loss 0.224, Val loss 0.348
  + Ep 4 (Step 002000): Train loss 0.119, Val loss 0.341
  + Ep 4 (Step 002200): Train loss 0.173, Val loss 0.337
  + **[Epoch 4] mean Train loss 0.164, mean Val loss 0.189**
  + Training accuracy: 92.19% | Validation accuracy: 85.94%
  + Ep 5 (Step 002400): Train loss 0.129, Val loss 0.330
  + Ep 5 (Step 002600): Train loss 0.115, Val loss 0.334
  + Ep 5 (Step 002800): Train loss 0.126, Val loss 0.320
  + **[Epoch 5] mean Train loss 0.129, mean Val loss 0.156**
  + Training accuracy: 96.88% | Validation accuracy: 85.94%
  + Ep 6 (Step 003000): Train loss 0.073, Val loss 0.313
  + Ep 6 (Step 003200): Train loss 0.145, Val loss 0.319
  + Ep 6 (Step 003400): Train loss 0.085, Val loss 0.315
  + **[Epoch 6] mean Train loss 0.112, mean Val loss 0.141**
  + Training accuracy: 93.75% | Validation accuracy: 87.50%
  + Ep 7 (Step 003600): Train loss 0.132, Val loss 0.306
  + Ep 7 (Step 003800): Train loss 0.120, Val loss 0.306
  + **[Epoch 7] mean Train loss 0.100, mean Val loss 0.132**
  + Training accuracy: 95.31% | Validation accuracy: 85.94%
  + Ep 8 (Step 004000): Train loss 0.035, Val loss 0.288
  + Ep 8 (Step 004200): Train loss 0.032, Val loss 0.269
  + Ep 8 (Step 004400): Train loss 0.039, Val loss 0.271
  + **[Epoch 8] mean Train loss 0.097, mean Val loss 0.132**
  + Training accuracy: 96.88% | Validation accuracy: 87.50%
  + Ep 9 (Step 004600): Train loss 0.078, Val loss 0.261
  + Ep 9 (Step 004800): Train loss 0.101, Val loss 0.246
  + Ep 9 (Step 005000): Train loss 0.050, Val loss 0.238
  + **[Epoch 9] mean Train loss 0.075, mean Val loss 0.107**
  + Training accuracy: 98.44% | Validation accuracy: 87.50%
  + Ep 10 (Step 005200): Train loss 0.027, Val loss 0.233
  + Ep 10 (Step 005400): Train loss 0.028, Val loss 0.221
  + Ep 10 (Step 005600): Train loss 0.043, Val loss 0.214
  + **[Epoch 10] mean Train loss 0.066, mean Val loss 0.099**
  + Training accuracy: 100.00% | Validation accuracy: 87.50%
  + Ep 11 (Step 005800): Train loss 0.082, Val loss 0.219
  + Ep 11 (Step 006000): Train loss 0.031, Val loss 0.213
  + Ep 11 (Step 006200): Train loss 0.059, Val loss 0.189
  + **[Epoch 11] mean Train loss 0.068, mean Val loss 0.106**
  + Training accuracy: 98.44% | Validation accuracy: 89.06%
  + Ep 12 (Step 006400): Train loss 0.039, Val loss 0.189
  + Ep 12 (Step 006600): Train loss 0.070, Val loss 0.196
  + Ep 12 (Step 006800): Train loss 0.013, Val loss 0.163
  + **[Epoch 12] mean Train loss 0.056, mean Val loss 0.092**
  + Training accuracy: 98.44% | Validation accuracy: 90.62%
  + Ep 13 (Step 007000): Train loss 0.010, Val loss 0.178
  + Ep 13 (Step 007200): Train loss 0.040, Val loss 0.162
  + Ep 13 (Step 007400): Train loss 0.179, Val loss 0.155
  + **[Epoch 13] mean Train loss 0.050, mean Val loss 0.087**
  + Training accuracy: 95.31% | Validation accuracy: 92.19%
  + Ep 14 (Step 007600): Train loss 0.021, Val loss 0.173
  + Ep 14 (Step 007800): Train loss 0.027, Val loss 0.155
  + **[Epoch 14] mean Train loss 0.046, mean Val loss 0.085**
  + Training accuracy: 100.00% | Validation accuracy: 93.75%
  + Ep 15 (Step 008000): Train loss 0.158, Val loss 0.143
  + Ep 15 (Step 008200): Train loss 0.021, Val loss 0.144
  + Ep 15 (Step 008400): Train loss 0.116, Val loss 0.130
  + **[Epoch 15] mean Train loss 0.041, mean Val loss 0.082**
  + Training accuracy: 98.44% | Validation accuracy: 93.75%
  + Ep 16 (Step 008600): Train loss 0.115, Val loss 0.134
  + Ep 16 (Step 008800): Train loss 0.018, Val loss 0.127
  + Ep 16 (Step 009000): Train loss 0.024, Val loss 0.129
  + **[Epoch 16] mean Train loss 0.038, mean Val loss 0.081**
  + Training accuracy: 100.00% | Validation accuracy: 93.75%
  + Ep 17 (Step 009200): Train loss 0.014, Val loss 0.118
  + Ep 17 (Step 009400): Train loss 0.051, Val loss 0.114
  + Ep 17 (Step 009600): Train loss 0.003, Val loss 0.104
  + **[Epoch 17] mean Train loss 0.033, mean Val loss 0.075**
  + Training accuracy: 100.00% | Validation accuracy: 93.75%
  + Ep 18 (Step 009800): Train loss 0.023, Val loss 0.113
  + Ep 18 (Step 010000): Train loss 0.024, Val loss 0.111
  + Ep 18 (Step 010200): Train loss 0.028, Val loss 0.105
  + **[Epoch 18] mean Train loss 0.035, mean Val loss 0.085**
  + Training accuracy: 100.00% | Validation accuracy: 93.75%
  + Ep 19 (Step 010400): Train loss 0.035, Val loss 0.109
  + Ep 19 (Step 010600): Train loss 0.005, Val loss 0.094
  + Ep 19 (Step 010800): Train loss 0.084, Val loss 0.101
  + **[Epoch 19] mean Train loss 0.027, mean Val loss 0.075**
  + Training accuracy: 98.44% | Validation accuracy: 93.75%
  + Ep 20 (Step 011000): Train loss 0.009, Val loss 0.097
  + Ep 20 (Step 011200): Train loss 0.023, Val loss 0.092
  + Ep 20 (Step 011400): Train loss 0.009, Val loss 0.100
  + **[Epoch 20] mean Train loss 0.027, mean Val loss 0.079**
  + Training accuracy: 98.44% | Validation accuracy: 93.75%
  + Early stopping: 3 epoch boyunca anlamlı iyileşme yok. En iyi Val loss 0.0755 (epoch 17).
  + Best weights geri yüklendi.
  + Training completed in 421.66 minutes.

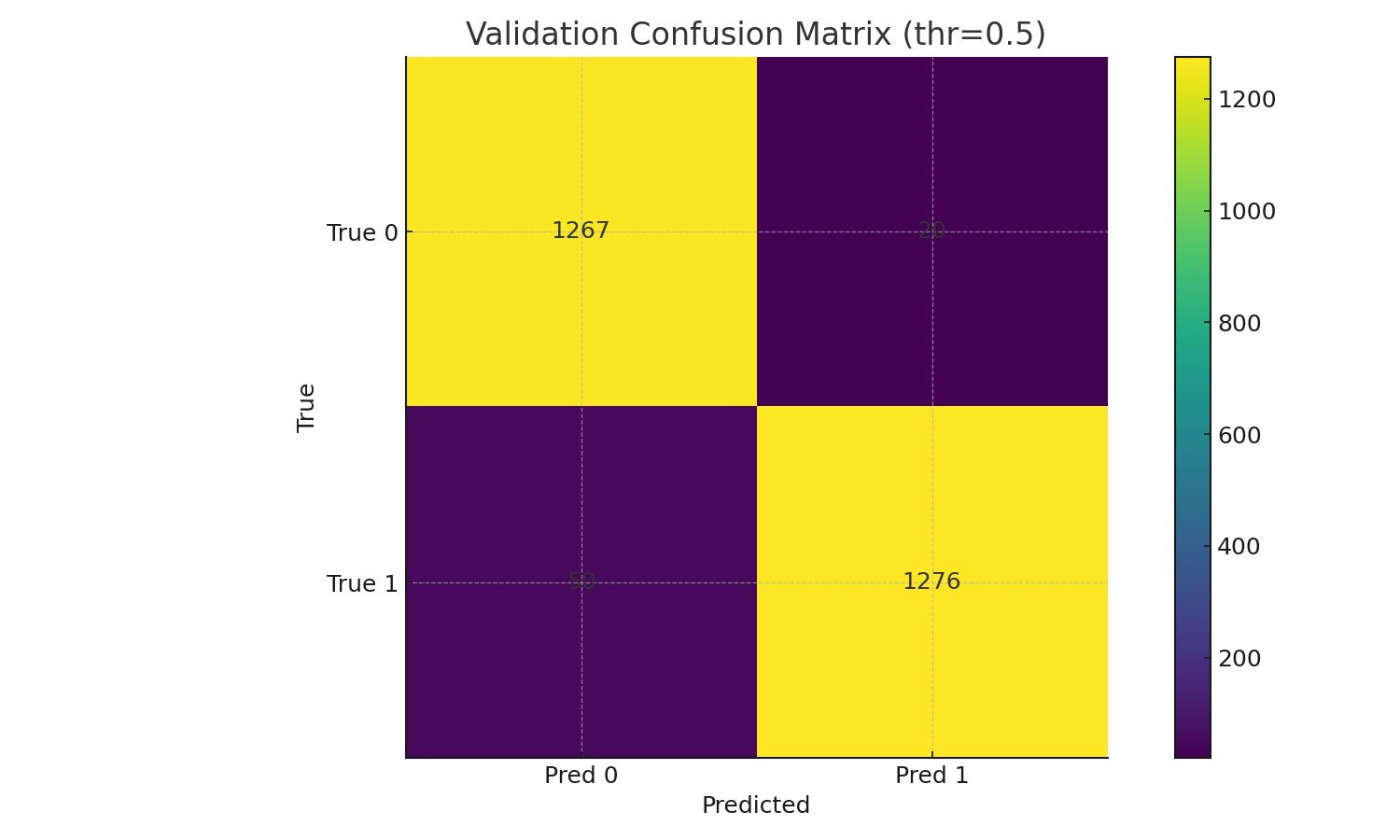
Maksimum epoch sayısı 20 olarak belirlenmiştir. 17.epochta en iyi sonuç alınmış ve zaten maksimum olan 20.epochta eğitim bitecekken eğitime başlamadan önce erken durdurma ayalarında 3 epochta 0.001 iyileşme yoksa kesileceği için 20.epochta early stopping’den dolayı eğitim tamamlanmıştır. Buradaki değerler küçük batchler ile ölçülüğünden tam sonuç vermeyebilir. O yüzden eğitim bittikten sonra hem tüm validation verisinde hem de test verisinde ölçümler yapılmış ve sonuçlar aşağıdaki gibi olmuştur.



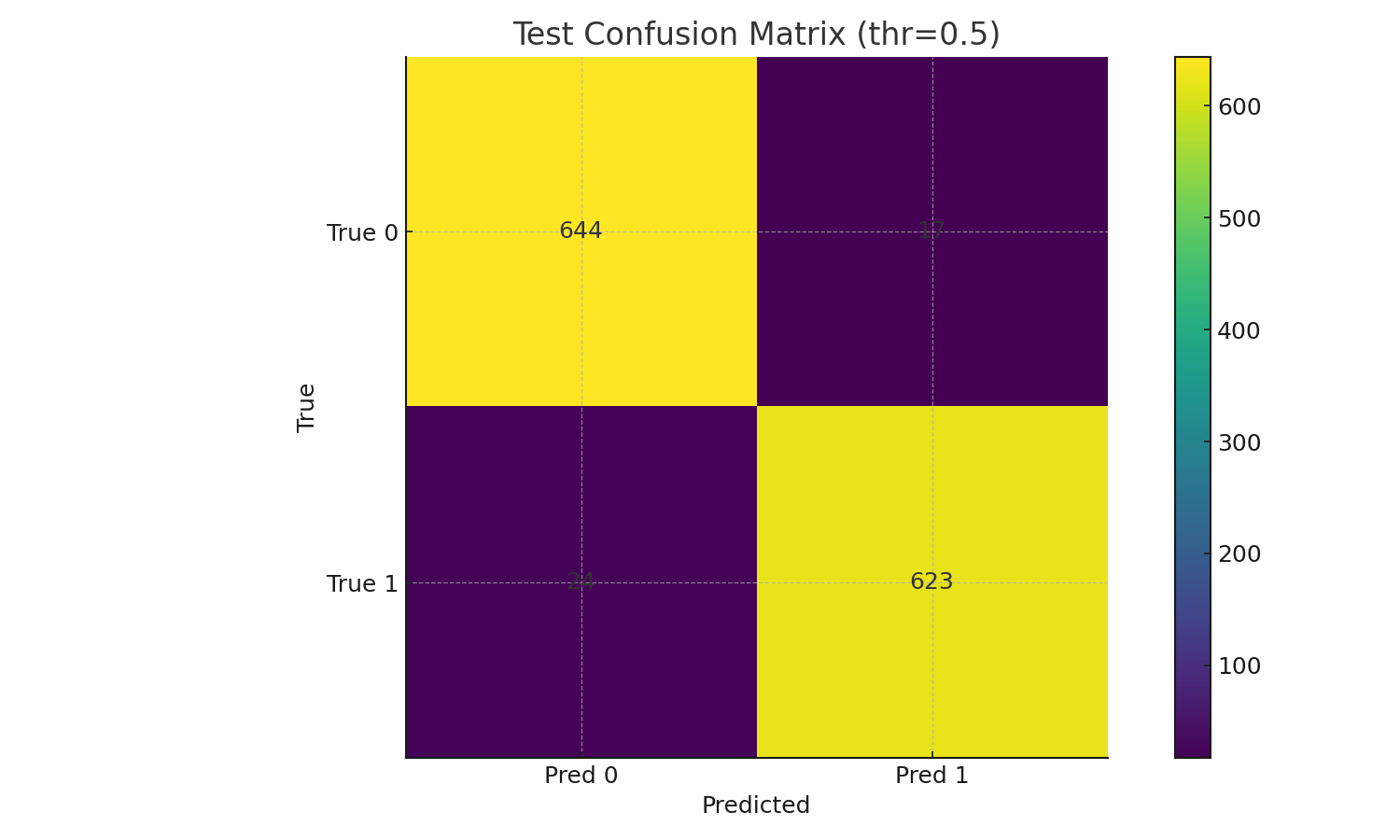
2.Eğitim Validasyon Metrikleri



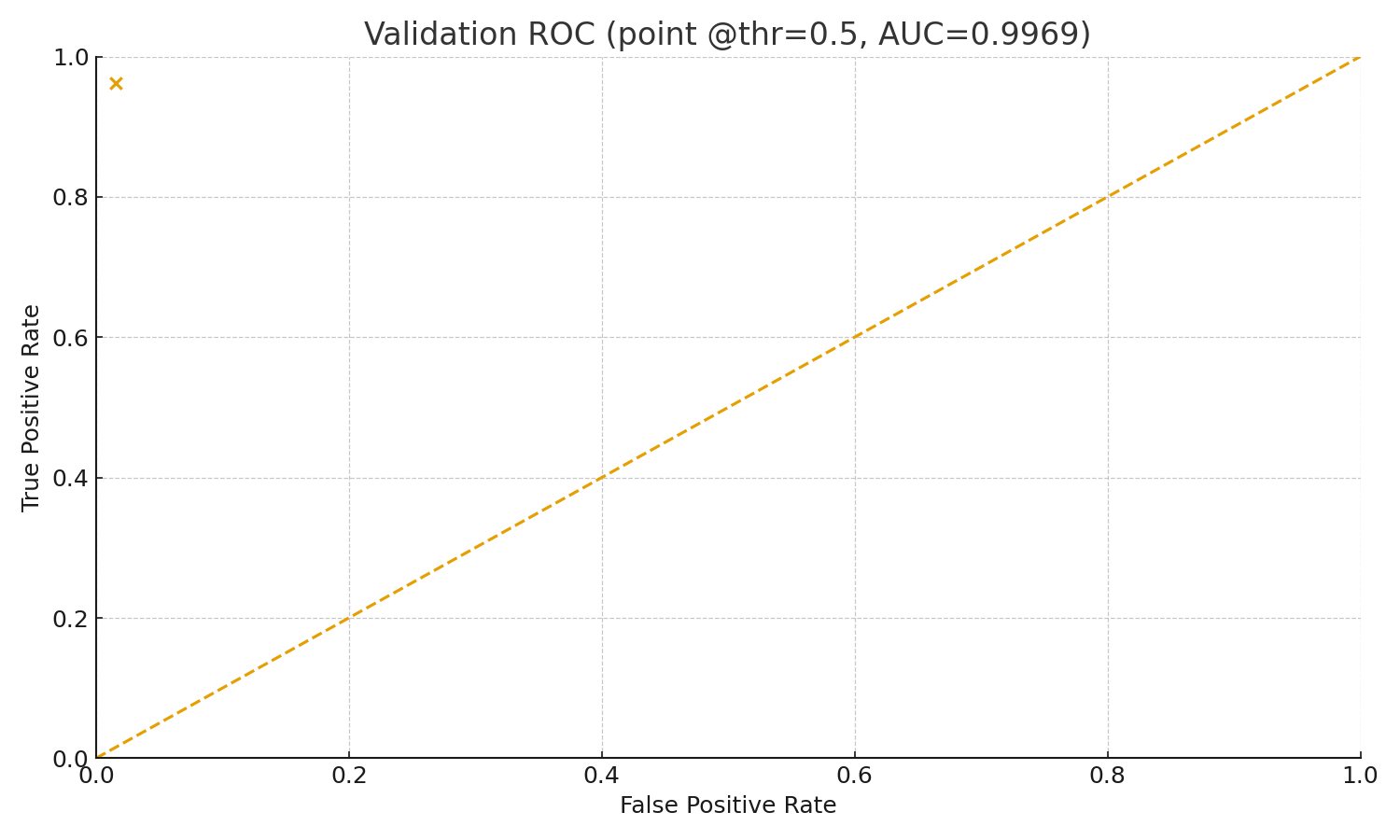
2.Eğitim Test Metrikleri



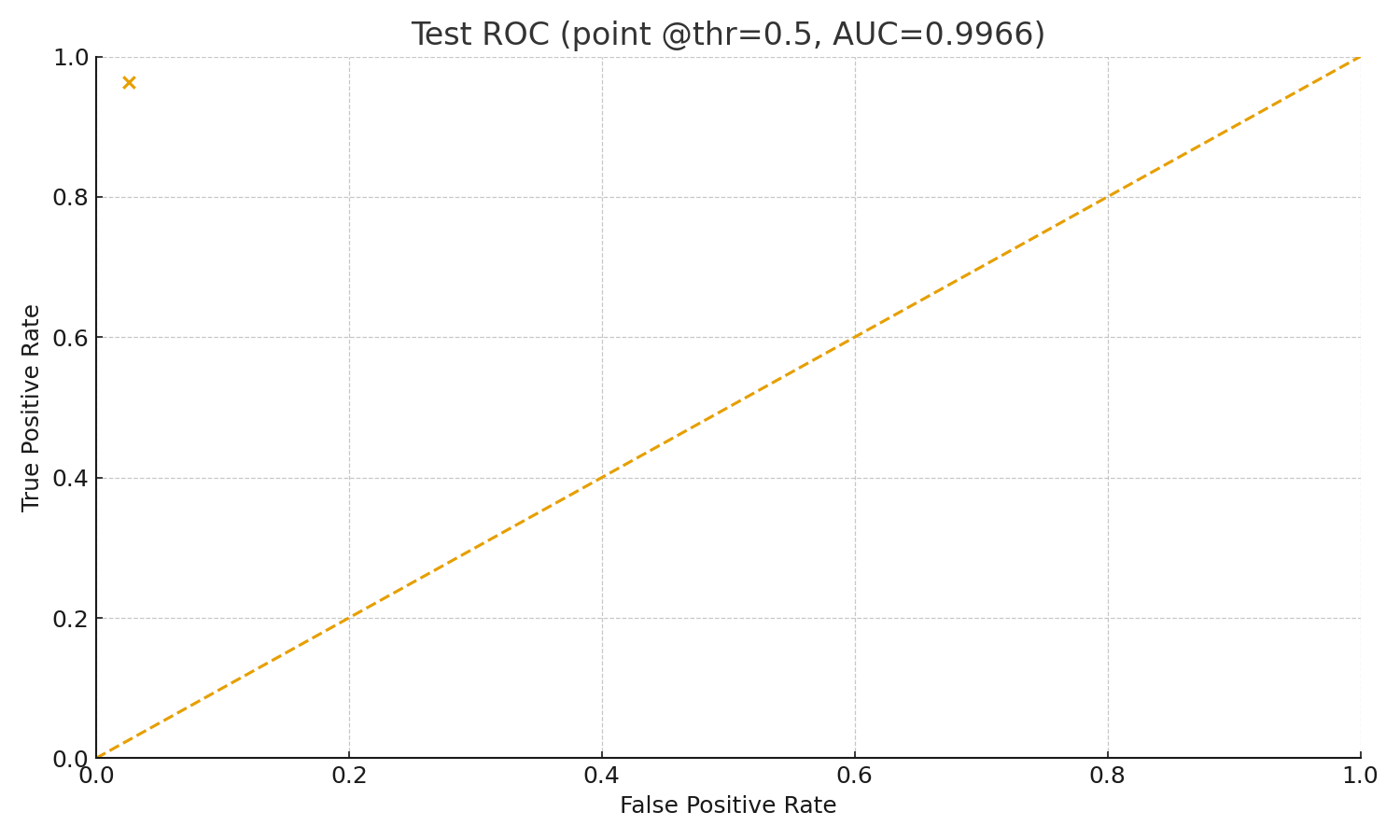
2.Eğitim Validasyon Confusion Matrix



2.Eğitim Test Confusion Matrix

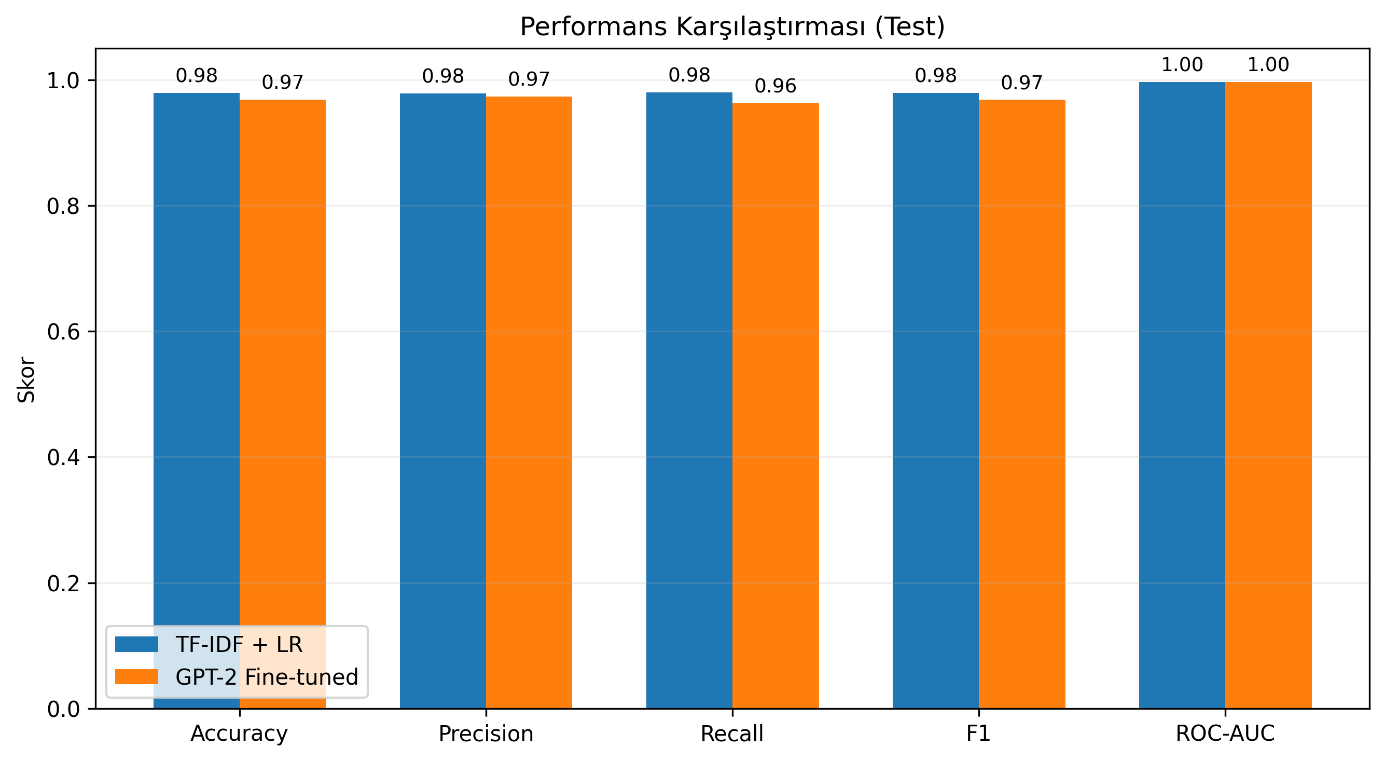


2.Eğitim Validasyon ROC



2.Eğitim Test ROC

Eğitim sonuçları incelendiğinde modelin başarımında ciddi oranda artış yaşandığı görülmektedir. Geleneksel NLP yöntemlerinden TF-IDF+Logistic Regression ile karşılaştırma yapmak için veri seti aynı veri ön işlemeden geçirilmiş ve veriler bu yöntemle eğitilmiştir. Sonuçlar aşağıda görülmektedir.



TF-IDF+LR ile GPT-2 Fine Tune Karşılaştırması

TF-IDF+LR eğitim sonuçları gpt-2 fine tune dan biraz daha iyi çıktığı görülmektedir. Sonuçları incelediğimde;

* + GPT-2 fine tune ederken 1024 tokendan uzun mailleri baş kısmını alıp geri kalan atılmıştır. Ancak TF-IDF+LR tüm mail setinden öğrenme yaptı. Literatüre bakıldığında bunun yerine sliding-window/Multiple Insantace Learning yapıları kullanabilirdi.
  + GPT-2 fine tune ederken bir tane daha transformer bloğu açıp eğitim yapıp sonuçlar gözlemlenebilirdi. Eğitim setinde az veri bulunduğundan belki daha fazla eğitim yapılması gerekiyordu.
  + Ancak iki eğitimi karşılaştırıldığında biri 421 dakika sürerken diğeri 5dk sürmüştür.

Başta yapılan literatür taramasında bulunan çalışmalar ile proje kapsamında fine tune edilen gpt-2 small modeline bakıldığında modelin başarımının birçok çalışmaya göre daha iyi olduğunu görülmüştür.

1. **Kaynak Kodları**

import pandas as pd

import re

import warnings

import html

import tiktoken

import time

from bs4 import BeautifulSoup, MarkupResemblesLocatorWarning

from ftfy import fix\_text

warnings.filterwarnings("ignore", category=MarkupResemblesLocatorWarning)

warnings.simplefilter(action="ignore", category=pd.errors.SettingWithCopyWarning)

class PhishingEmailPreprocessor:

"""

Veri ön işleme için yazılan sınıf.

"""

def \_\_init\_\_(self, csv\_path:str, train\_frac:float, validation\_frac:float):

self.df = None

self.train\_df = None

self.validation\_df = None

self.test\_df = None

self.train\_frac = train\_frac

self.validation\_frac = validation\_frac

self.csv\_path = csv\_path

self.tokenizer = tiktoken.get\_encoding("gpt2")

self.pad\_token\_id = 50256

self.max\_len = 1024

self.load\_and\_prepare(self.csv\_path)

def strip\_html(self,text: str) -> str:

if not isinstance(text, str):

return ""

text = html.unescape(text)

soup = BeautifulSoup(text, "html.parser")

return soup.get\_text(separator=" ")

def normalize\_whitespace(self,text: str) -> str:

if not isinstance(text, str):

return ""

text = re.sub(r"\s+", " ", text)

return text.strip()

def fix\_unicode(self,text: str) -> str:

if not isinstance(text, str):

return ""

return fix\_text(text)

def create\_balanced\_dataset(self):

num\_spam = self.df[self.df["Email Type"] == 1].shape[0]

ham\_subset = self.df[self.df["Email Type"] == 0].sample(num\_spam, random\_state=123)

self.df = pd.concat([ham\_subset, self.df[self.df["Email Type"] == 1]])

def load\_and\_prepare(self,csv\_path: str):

self.df = pd.read\_csv(csv\_path, sep=",", header=0, quotechar='"')

self.df = self.df.dropna(subset=["Email Text", "Email Type"])

self.df = self.df[self.df["Email Text"].astype(str).str.strip() != ""]

self.df["Email Text"] = self.df["Email Text"].apply(self.fix\_unicode)

self.df["Email Text"] = self.df["Email Text"].apply(self.strip\_html)

self.df["Email Text"] = self.df["Email Text"].apply(self.normalize\_whitespace)

self.df = self.df[self.df["Email Text"].str.len() > 0]

self.df = self.df.drop\_duplicates(subset=["Email Text"]).copy()

self.df["Email Type"] = self.df["Email Type"].map({"Safe Email": 0, "Phishing Email": 1})

self.df = self.df.dropna(subset=["Email Type"])

self.df["Email Type"] = self.df["Email Type"].astype(int)

self.create\_balanced\_dataset()

self.random\_split()

self.train\_df["Token"] = self.train\_df["Email Text"].apply(self.tokenize)

self.validation\_df["Token"] = self.validation\_df["Email Text"].apply(self.tokenize)

self.test\_df["Token"] = self.test\_df["Email Text"].apply(self.tokenize)

def random\_split(self):

self.df = self.df.sample(frac=1, random\_state=123).reset\_index(drop=True)

train\_end = int(len(self.df) \* self.train\_frac)

validation\_end = train\_end + int(len(self.df) \* self.validation\_frac)

self.train\_df = self.df[:train\_end]

self.validation\_df = self.df[train\_end:validation\_end]

self.test\_df = self.df[validation\_end:]

def tokenize(self,text):

tokens = self.tokenizer.encode(text)

return tokens

def get\_dfs(self) -> pd.DataFrame:

self.train\_df = self.train\_df.drop(columns=["Unnamed: 0"])

self.validation\_df = self.validation\_df.drop(columns=["Unnamed: 0"])

self.test\_df = self.test\_df.drop(columns=["Unnamed: 0"])

return self.train\_df, self.validation\_df, self.test\_df

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

preprocessor = PhishingEmailPreprocessor("C:\\Users\\user\\Desktop\\Phishing\_Email.csv\\Phishing\_Email.csv",0.7,0.2)

train\_df, validation\_df, test\_df = preprocessor.get\_dfs()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import torch

from torch.utils.data import Dataset, DataLoader

"""

Dataframeden DataLoadera dönüşüm için aşağıdaki 2 sınıf kullanılmıştır.

"""

class EmailDataset(Dataset):

def \_\_init\_\_(self, df):

self.df = df.reset\_index(drop=True)

def \_\_len\_\_(self):

return len(self.df)

def \_\_getitem\_\_(self, idx):

tokens = self.df.loc[idx, "Token"]

label = int(self.df.loc[idx, "Email Type"])

return tokens, label

class EmailDataLoader:

def \_\_init\_\_(self, train\_df, val\_df, test\_df, batch\_size=8, num\_workers=0, seed=123):

torch.manual\_seed(seed)

self.batch\_size = batch\_size

self.num\_workers = num\_workers

self.train\_dataset = EmailDataset(train\_df)

self.val\_dataset = EmailDataset(val\_df)

self.test\_dataset = EmailDataset(test\_df)

#Bir batchdeki maksimum token uzunluğunu bulup batch padding yapmak için yazılmıştır.

def make\_collate\_fn(self, pad\_id=preprocessor.pad\_token\_id, max\_len=None):

def collate(batch):

xs, ys = zip(\*batch)

xs\_list = [x.tolist() if isinstance(x, torch.Tensor) else x for x in xs]

if max\_len is not None:

xs\_list = [x[:max\_len] for x in xs\_list]

maxlen = max(len(x) for x in xs\_list)

X = [x + [pad\_id] \* (maxlen - len(x)) for x in xs\_list]

Y = list(ys)

X = torch.tensor(X, dtype=torch.long)

Y = torch.tensor(Y, dtype=torch.long)

return X, Y

return collate

def get\_loaders(self):

collate = self.make\_collate\_fn(pad\_id=preprocessor.pad\_token\_id, max\_len=preprocessor.max\_len)

train\_loader = DataLoader(

dataset=self.train\_dataset,

batch\_size=self.batch\_size,

shuffle=True,

num\_workers=self.num\_workers,

drop\_last=True,

collate\_fn=collate,

)

val\_loader = DataLoader(

dataset=self.val\_dataset,

batch\_size=self.batch\_size,

shuffle=False,

num\_workers=self.num\_workers,

drop\_last=False,

collate\_fn=collate,

)

test\_loader = DataLoader(

dataset=self.test\_dataset,

batch\_size=self.batch\_size,

shuffle=False,

num\_workers=self.num\_workers,

drop\_last=False,

collate\_fn=collate,

)

return train\_loader, val\_loader, test\_loader

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

data\_module = EmailDataLoader(train\_df, validation\_df, test\_df, batch\_size=16)

train\_loader, val\_loader, test\_loader = data\_module.get\_loaders()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import torch.nn as nn

import torch

class MultiHeadAttention(nn.Module):

"""

Attention Sınıfı.

"""

def \_\_init\_\_(self, d\_in, d\_out, context\_length, dropout, num\_heads, qkv\_bias=False):

super().\_\_init\_\_()

assert (d\_out % num\_heads == 0), \

self.d\_out = d\_out

self.num\_heads = num\_heads

self.head\_dim = d\_out // num\_heads #

self.W\_query = nn.Linear(d\_in, d\_out, bias=qkv\_bias)

self.W\_key = nn.Linear(d\_in, d\_out, bias=qkv\_bias)

self.W\_value = nn.Linear(d\_in, d\_out, bias=qkv\_bias)

self.out\_proj = nn.Linear(d\_out, d\_out)

self.dropout = nn.Dropout(dropout)

self.register\_buffer(

"mask",

torch.triu(torch.ones(context\_length, context\_length),

diagonal=1)

)

def forward(self, x):

b, num\_tokens, d\_in = x.shape

keys = self.W\_key(x)

queries = self.W\_query(x)

values = self.W\_value(x)

keys = keys.view(b, num\_tokens, self.num\_heads, self.head\_dim)

values = values.view(b, num\_tokens, self.num\_heads, self.head\_dim)

queries = queries.view(b, num\_tokens, self.num\_heads, self.head\_dim)

keys = keys.transpose(1, 2)

queries = queries.transpose(1, 2)

values = values.transpose(1, 2)

attn\_scores = queries @ keys.transpose(2, 3)

mask\_bool = self.mask.bool()[:num\_tokens, :num\_tokens]

attn\_scores.masked\_fill\_(mask\_bool, -torch.inf)

attn\_weights = torch.softmax(attn\_scores / keys.shape[-1]\*\*0.5, dim=-1)

attn\_weights = self.dropout(attn\_weights)

context\_vec = (attn\_weights @ values).transpose(1, 2)

context\_vec = context\_vec.contiguous().view(b, num\_tokens, self.d\_out)

context\_vec = self.out\_proj(context\_vec) # optional projection

return context\_vec

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import torch.nn as nn

import torch

class GELU(nn.Module):

"""

Aktivasyon fonksiyonu için yazılmış sınıf.

"""

def \_\_init\_\_(self):

super().\_\_init\_\_()

def forward(self, x):

return 0.5 \* x \* (1 + torch.tanh(

torch.sqrt(torch.tensor(2.0 / torch.pi)) \*

(x + 0.044715 \* torch.pow(x, 3))

))

class FeedForward(nn.Module):

def \_\_init\_\_(self, cfg):

super().\_\_init\_\_()

self.layers = nn.Sequential(

nn.Linear(cfg["emb\_dim"], 4 \* cfg["emb\_dim"]),

GELU(),

nn.Linear(4 \* cfg["emb\_dim"], cfg["emb\_dim"]),

)

def forward(self, x):

return self.layers(x)

class LayerNorm(nn.Module):

def \_\_init\_\_(self, emb\_dim):

super().\_\_init\_\_()

self.eps = 1e-5

self.scale = nn.Parameter(torch.ones(emb\_dim))

self.shift = nn.Parameter(torch.zeros(emb\_dim))

def forward(self, x):

mean = x.mean(dim=-1, keepdim=True)

var = x.var(dim=-1, keepdim=True, unbiased=False)

norm\_x = (x - mean) / torch.sqrt(var + self.eps)

return self.scale \* norm\_x + self.shift

class TransformerBlock(nn.Module):

def \_\_init\_\_(self, cfg):

super().\_\_init\_\_()

self.att = MultiHeadAttention(

d\_in=cfg["emb\_dim"],

d\_out=cfg["emb\_dim"],

context\_length=cfg["context\_length"],

num\_heads=cfg["n\_heads"],

dropout=cfg["drop\_rate"],

qkv\_bias=cfg["qkv\_bias"])

self.ff = FeedForward(cfg)

self.norm1 = LayerNorm(cfg["emb\_dim"])

self.norm2 = LayerNorm(cfg["emb\_dim"])

self.drop\_shortcut = nn.Dropout(cfg["drop\_rate"])

def forward(self, x):

shortcut = x

x = self.norm1(x)

x = self.att(x)

x = self.drop\_shortcut(x)

x = x + shortcut

shortcut = x

x = self.norm2(x)

x = self.ff(x)

x = self.drop\_shortcut(x)

x = x + shortcut

return x

class GPTModel(nn.Module):

"""

Kullanılacak modelimiz.

"""

def \_\_init\_\_(self, cfg):

super().\_\_init\_\_()

self.tok\_emb = nn.Embedding(cfg["vocab\_size"], cfg["emb\_dim"])

self.pos\_emb = nn.Embedding(cfg["context\_length"], cfg["emb\_dim"])

self.drop\_emb = nn.Dropout(cfg["drop\_rate"])

self.trf\_blocks = nn.Sequential(

\*[TransformerBlock(cfg) for \_ in range(cfg["n\_layers"])])

self.final\_norm = LayerNorm(cfg["emb\_dim"])

self.out\_head = nn.Linear(

cfg["emb\_dim"], cfg["vocab\_size"], bias=False

)

def forward(self, in\_idx):

batch\_size, seq\_len = in\_idx.shape

tok\_embeds = self.tok\_emb(in\_idx)

pos\_embeds = self.pos\_emb(torch.arange(seq\_len, device=in\_idx.device))

x = tok\_embeds + pos\_embeds # Shape [batch\_size, num\_tokens, emb\_size]

x = self.drop\_emb(x)

x = self.trf\_blocks(x)

x = self.final\_norm(x)

logits = self.out\_head(x)

return logits

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#Modeli huggingface'den indirip config bilgisi ile yükledim.

file\_name = "D:\\llm\\model\\gpt2-small-124M.pth"

GPT\_CONFIG\_124M = {

"vocab\_size": 50257, # Vocabulary size

"context\_length": 1024, # Shortened context length (orig: 1024)

"emb\_dim": 768, # Embedding dimension

"n\_heads": 12, # Number of attention heads

"n\_layers": 12, # Number of layers

"drop\_rate": 0.1, # Dropout rate

"qkv\_bias": True # Query-key-value bias

}

torch.manual\_seed(123)

model = GPTModel(GPT\_CONFIG\_124M)

model.load\_state\_dict(torch.load(file\_name, weights\_only=True))

model.eval()

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

model.to(device)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#Eğitime başlamadan önce tüm ağırlıkları dondurdum.

for param in model.parameters():

param.requires\_grad = False

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

torch.manual\_seed(123)

#Modelin son kısmını 2 çıkışlı network ile değiştirdim.

num\_classes = 2

model.out\_head = torch.nn.Linear(in\_features=GPT\_CONFIG\_124M["emb\_dim"], out\_features=num\_classes)

model.out\_head = model.out\_head.to(device)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#Son transformer block ve finalLayerNorm kısmını eğitim amaçlı açtım.

for param in model.trf\_blocks[-1].parameters():

param.requires\_grad = True

for param in model.final\_norm.parameters():

param.requires\_grad = True

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PAD\_ID = 50256

#Modelde attentionda PAD mask olmadığı için en son valid token almak için yazıldı.

def last\_valid\_logits(xb, model, device, pad\_id=PAD\_ID):

xb = xb.to(device)

last\_idx = (xb != pad\_id).sum(dim=1) - 1

logits\_all = model(xb) # [B, T, C]

return logits\_all[torch.arange(xb.size(0), device=device), last\_idx, :] # [B, C]

#cross\_entropy loss kullandım.

def calc\_loss\_batch(input\_batch, target\_batch, model, device):

input\_batch, target\_batch = input\_batch.to(device), target\_batch.to(device)

logits = last\_valid\_logits(input\_batch, model, device, PAD\_ID)

loss = torch.nn.functional.cross\_entropy(logits, target\_batch)

return loss

def calc\_loss\_loader(data\_loader, model, device, num\_batches=None):

total\_loss = 0.

if len(data\_loader) == 0:

return float("nan")

elif num\_batches is None:

num\_batches = len(data\_loader)

else:

num\_batches = min(num\_batches, len(data\_loader))

for i, (input\_batch, target\_batch) in enumerate(data\_loader):

if i < num\_batches:

loss = calc\_loss\_batch(input\_batch, target\_batch, model, device)

total\_loss += loss.item()

else:

break

return total\_loss / num\_batches

def calc\_accuracy\_loader(data\_loader, model, device, num\_batches=None):

model.eval()

correct\_predictions, num\_examples = 0, 0

if num\_batches is None:

num\_batches = len(data\_loader)

else:

num\_batches = min(num\_batches, len(data\_loader))

for i, (input\_batch, target\_batch) in enumerate(data\_loader):

if i < num\_batches:

input\_batch, target\_batch = input\_batch.to(device), target\_batch.to(device)

with torch.no\_grad():

logits = last\_valid\_logits(input\_batch, model, device, PAD\_ID)

predicted\_labels = torch.argmax(logits, dim=-1)

num\_examples += predicted\_labels.shape[0]

correct\_predictions += (predicted\_labels == target\_batch).sum().item()

else:

break

return correct\_predictions / num\_examples

def evaluate\_model(model, train\_loader, val\_loader, device, eval\_iter):

model.eval()

with torch.no\_grad():

train\_loss = calc\_loss\_loader(train\_loader, model, device, num\_batches=eval\_iter)

val\_loss = calc\_loss\_loader(val\_loader, model, device, num\_batches=eval\_iter)

model.train()

return train\_loss, val\_loss

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import copy, math, torch

#Eğitim için early stopping ile kullanacağımız fonksiyon

def train\_classifier\_simplev2(

model, train\_loader, val\_loader, optimizer, device,

num\_epochs, eval\_freq, eval\_iter,

early\_stopping=True, patience=3, min\_delta=1e-3,

restore\_best\_weights=True, checkpoint\_path=None

):

# Takip listeleri

train\_losses, val\_losses, train\_accs, val\_accs = [], [], [], []

examples\_seen, global\_step = 0, -1

# Early stopping durum değişkenleri

best\_val\_loss = math.inf

best\_state\_dict = None

epochs\_no\_improve = 0

best\_epoch = -1

for epoch in range(num\_epochs):

model.train()

for input\_batch, target\_batch in train\_loader:

optimizer.zero\_grad()

loss = calc\_loss\_batch(input\_batch, target\_batch, model, device)

loss.backward()

optimizer.step()

examples\_seen += input\_batch.shape[0]

global\_step += 1

if global\_step % eval\_freq == 0 and global\_step > 0:

train\_loss, val\_loss = evaluate\_model(

model, train\_loader, val\_loader, device, eval\_iter=eval\_iter

)

train\_losses.append(train\_loss)

val\_losses.append(val\_loss)

print(f"Ep {epoch+1} (Step {global\_step:06d}): "

f"Train loss {train\_loss:.3f}, Val loss {val\_loss:.3f}")

epoch\_train\_loss, epoch\_val\_loss = evaluate\_model(

model, train\_loader, val\_loader, device, eval\_iter=None # tamamını tara

)

train\_accuracy = calc\_accuracy\_loader(train\_loader, model, device, num\_batches=eval\_iter)

val\_accuracy = calc\_accuracy\_loader(val\_loader, model, device, num\_batches=eval\_iter)

print(f"[Epoch {epoch+1}] mean Train loss {epoch\_train\_loss:.3f}, mean Val loss {epoch\_val\_loss:.3f}")

print(f"Training accuracy: {train\_accuracy\*100:.2f}% | Validation accuracy: {val\_accuracy\*100:.2f}%")

train\_accs.append(train\_accuracy)

val\_accs.append(val\_accuracy)

# ---- EARLY STOPPING KONTROLÜ ----

if early\_stopping:

improved = (best\_val\_loss - epoch\_val\_loss) > min\_delta

if improved:

best\_val\_loss = epoch\_val\_loss

best\_epoch = epoch + 1

epochs\_no\_improve = 0

if restore\_best\_weights:

best\_state\_dict = copy.deepcopy(model.state\_dict())

if checkpoint\_path: # opsiyonel diske kaydet

torch.save(model.state\_dict(), checkpoint\_path)

else:

epochs\_no\_improve += 1

if epochs\_no\_improve >= patience:

print(f"Early stopping: {patience} epoch boyunca anlamlı iyileşme yok. "

f"En iyi Val loss {best\_val\_loss:.4f} (epoch {best\_epoch}).")

if restore\_best\_weights and best\_state\_dict is not None:

model.load\_state\_dict(best\_state\_dict)

print("Best weights geri yüklendi.")

break

# ---- /EARLY STOPPING ----

return train\_losses, val\_losses, train\_accs, val\_accs, examples\_seen

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import time

start\_time = time.time()

torch.manual\_seed(123)

#Optimizer olarak AdamW kullandım ve lr ile weight\_decay değerlerini derste gördüğümüz örneğe göre biraz düşürdüm. (1.Eğitimde başarılı olmadı o değerler)

optimizer = torch.optim.AdamW(model.parameters(), lr=1e-5, weight\_decay=0.01)

num\_epochs = 20

train\_losses, val\_losses, train\_accs, val\_accs, examples\_seen = train\_classifier\_simplev2(

model, train\_loader, val\_loader, optimizer, device,

num\_epochs=num\_epochs,

eval\_freq=200, # step içinde kısa örneklemle rapor

eval\_iter=4, # step içi değerlendirmede kaç batch taransın

early\_stopping=True,

patience=3, # 3 epoch iyileşme yoksa dur

min\_delta=1e-3, # val loss en az bu kadar düşerse "iyileşme" say

restore\_best\_weights=True,

checkpoint\_path="phishing\_classifier2\_.pth" # istersen "best.pt" ver

)

end\_time = time.time()

execution\_time\_minutes = (end\_time - start\_time) / 60

print(f"Training completed in {execution\_time\_minutes:.2f} minutes.")

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

torch.save(model.state\_dict(), "phishing\_classifier2.pth")

import pickle

metrics = { "train\_losses": train\_losses, "val\_losses": val\_losses, "train\_accs": train\_accs, "val\_accs": val\_accs, "examples\_seen": examples\_seen }

with open("metrics2.pkl", "wb") as f:

pickle.dump(metrics, f, protocol=pickle.HIGHEST\_PROTOCOL)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import torch

import numpy as np

@torch.no\_grad()

def eval\_full(val\_loader, model, device, pad\_id=50256):

model.eval()

tp = fp = tn = fn = 0

all\_probs = []

all\_y = []

for xb, yb in val\_loader:

xb = xb.to(device, non\_blocking=True)

yb = yb.to(device, non\_blocking=True)

# Son geçerli token'ın indeksini bul

last\_idx = (xb != pad\_id).sum(dim=1) - 1

# Logits'i o pozisyondan al

logits\_all = model(xb) # [B, T, C]

logits = logits\_all[torch.arange(xb.size(0), device=device), last\_idx, :] # [B, C]

preds = torch.argmax(logits, dim=-1) # [B]

probs = torch.softmax(logits, dim=-1)[:, 1] # positive sınıf olasılığı

# Confusion matrix bileşenleri

tp += ((preds == 1) & (yb == 1)).sum().item()

tn += ((preds == 0) & (yb == 0)).sum().item()

fp += ((preds == 1) & (yb == 0)).sum().item()

fn += ((preds == 0) & (yb == 1)).sum().item()

all\_probs.append(probs.detach().cpu())

all\_y.append(yb.detach().cpu())

# Temel metrikler

total = tp + tn + fp + fn

acc = (tp + tn) / max(1, total)

precision = tp / max(1, tp + fp)

recall = tp / max(1, tp + fn)

f1 = 2 \* precision \* recall / max(1e-12, (precision + recall))

# ROC-AUC (ranks yöntemi; sklearn yok)

y\_true = torch.cat(all\_y).numpy()

scores = torch.cat(all\_probs).numpy()

pos = (y\_true == 1)

n\_pos = int(pos.sum())

n\_neg = len(y\_true) - n\_pos

auc = None

if n\_pos > 0 and n\_neg > 0:

order = np.argsort(scores)

ranks = np.empty\_like(order)

ranks[order] = np.arange(len(scores)) + 1

sum\_ranks\_pos = ranks[pos].sum()

auc = (sum\_ranks\_pos - n\_pos \* (n\_pos + 1) / 2) / (n\_pos \* n\_neg)

return {

"accuracy": acc,

"precision": precision,

"recall": recall,

"f1": f1,

"roc\_auc": auc,

"confusion\_matrix": {"tn": tn, "fp": fp, "fn": fn, "tp": tp},

}

metrics = eval\_full(val\_loader, model, device)

test\_metrics = eval\_full(test\_loader, model, device, pad\_id=50256)

print(metrics)

print(test\_metrics)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#TF-IDF vs fine-tuned GPT2-small

import pandas as pd

import numpy as np

from sklearn.pipeline import Pipeline

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score, confusion\_matrix, roc\_curve, auc

import matplotlib.pyplot as plt

X\_train = train\_df["Email Text"].astype(str).values

y\_train = train\_df["Email Type"].astype(int).values

X\_val = validation\_df["Email Text"].astype(str).values

y\_val = validation\_df["Email Type"].astype(int).values

X\_test = test\_df["Email Text"].astype(str).values

y\_test = test\_df["Email Type"].astype(int).values

pipe = Pipeline([

("tfidf", TfidfVectorizer(

lowercase=False,

strip\_accents="unicode",

sublinear\_tf=True

)),

("lr", LogisticRegression(

max\_iter=2000,

n\_jobs=None,

solver="liblinear"

))

])

param\_grid = {

"tfidf\_\_ngram\_range": [(1,1), (1,2)],

"tfidf\_\_min\_df": [2, 5],

"tfidf\_\_max\_features": [None, 100\_000],

"lr\_\_C": [0.5, 1.0, 2.0],

"lr\_\_penalty": ["l2"]

}

gs = GridSearchCV(

pipe,

param\_grid=param\_grid,

scoring="f1",

cv=3,

n\_jobs=-1,

verbose=1

)

gs.fit(X\_train, y\_train)

baseline = gs.best\_estimator\_

# === 3) Validation ve Test performansı ===

def evaluate(model, X, y):

proba = model.predict\_proba(X)[:,1]

pred = (proba >= 0.5).astype(int)

return {

"accuracy": accuracy\_score(y, pred),

"precision": precision\_score(y, pred, zero\_division=0),

"recall": recall\_score(y, pred, zero\_division=0),

"f1": f1\_score(y, pred, zero\_division=0),

"roc\_auc": roc\_auc\_score(y, proba),

"confusion\_matrix": confusion\_matrix(y, pred, labels=[0,1])

}

baseline\_val = evaluate(baseline, X\_val, y\_val)

baseline\_test = evaluate(baseline, X\_test, y\_test)

# GPT2 test veri setindeki sonuçlarım.

gpt2\_test = {

"accuracy": 0.9686544342507645,

"precision": 0.9734375,

"recall": 0.9629057187017002,

"f1": 0.9681429681429681,

"roc\_auc": 0.9965627462488338,

"confusion\_matrix": np.array([[644, 17],[24, 623]])

}

def plot\_compare(baseline\_metrics, gpt2\_metrics, title\_suffix="(Test Set)"):

labels = ["Accuracy","Precision","Recall","F1","ROC-AUC"]

base\_vals = [baseline\_metrics[k.lower()] for k in ["ACCURACY","PRECISION","RECALL","F1","ROC\_AUC"]]

gpt2\_vals = [gpt2\_metrics[k.lower()] for k in ["ACCURACY","PRECISION","RECALL","F1","ROC\_AUC"]]

x = np.arange(len(labels))

w = 0.35

plt.figure(figsize=(9,5))

b1 = plt.bar(x - w/2, base\_vals, width=w, label="TF-IDF + LR")

b2 = plt.bar(x + w/2, gpt2\_vals, width=w, label="GPT-2 Fine-tuned")

for bars in [b1, b2]:

for b in bars:

h = b.get\_height()

plt.text(b.get\_x() + b.get\_width()/2, h+0.01, f"{h:.2f}", ha="center", va="bottom", fontsize=9)

plt.ylim(0,1.05)

plt.xticks(x, labels)

plt.title(f"Performans Karşılaştırması {title\_suffix}")

plt.ylabel("Skor")

plt.legend()

plt.grid(axis="y", alpha=0.2)

plt.tight\_layout()

plt.show()

plot\_compare(baseline\_test, gpt2\_test, "(Test)")