UNIVERZITET CRNE GORE

ELEKTROTEHNIČKI FAKULTET

Danilo Vavić

AUTOMATIZOVANA KLASIFIKACIJA I ANALIZA MAIL PORUKA PRIMJENOM VJEŠTAČKE INTELIGENCIJE

-master rad-

Podgorica, 2025.

UNIVERZITET CRNE GORE

ELEKTROTEHNIČKI FAKULTET

Danilo Vavić

AUTOMATIZOVANA KLASIFIKACIJA I ANALIZA MAIL PORUKA PRIMJENOM VJEŠTAČKE INTELIGENCIJE

-master rad-

Podgorica, 2025.

PODACI I INFORMACIJE O STUDENTU

Ime i prezime:	Danilo Vavić
Datum i mjesto rođenja:	10.07.1998. godine, Cetinje, Crna gora
Naziv završenog osnovnog studijskog programa i godina završetka studija:	Studijski program Primijenjeno računarstvo, Elektrotehnički fakultet, Univerzitet Crne Gore, 240 ECTS, 2020. godine

INFORMACIJE O MASTER RADU

Naziv master studija:	Master studije primijenjenog računarstva
Naslov rada:	Automatizovana klasifikacija i analiza mail poruka primjenom vještačke inteligencije
Fakultet na kojem je rad odbranjen:	Elektrotehnički fakultet

UDK, OCJENA I ODBRANA MASTER RADA

Datum prijave magistarskog rada:	12.06.2025. godine
Datum sjednice Vijeća na kojoj je prihvaćena tema:	16.06.2025. godine
Komisija za ocjenu/odbranu rada:	Prof. dr Vesna Popović – Bugarin, ETF Podgorica, predsjednik, Prof. dr Nikola Žarić, ETF Podgorica, mentor, Doc. dr Miloš Brajović, ETF Podgorica, član
Mentor:	Prof. dr Nikola Žarić
Datum odbrane:	
Ime i prezime autora:	Danilo Vavić

Izjava o autorstvu

Potpisani Danilo Vavić

Broj indeksa/upisa 1051/22

Izjavljujem

da je master rad pod naslovom:

„AUTOMATIZOVANA KLASIFIKACIJA I ANALIZA MAIL PORUKA PRIMJENOM VJEŠTAČKE INTELIGENCIJE“

- rezultat sopstvenog istraživačkog rada,
- da predloženi master rad ni u cjelini ni u djelovima nije bio predložen za dobijanje bilo koje diplome prema studijskim programima drugih ustanova visokog obrazovanja,
- da su rezultati korektno navedeni, i
- da nijesam povrijedio/la autorska i druga prava intelektualne svojine koja pripadaju trećim licima.

U Podgorici, 24.11.2025. godine

Potpis magistranta

DV

ZAHVALNICA

Ovaj master rad posvećujem svojoj porodici, koja mi je tokom obrazovanja bila stalna podrška i oslonac. Hvala vam na razumijevanju, strpljenju i povjerenju koje ste mi pružali, kao i na vjeri u moje sposobnosti u svim fazama mog školovanja. Vaša podrška i ohrabrenje bili su važan dio svakog mog uspjeha. Takođe, želim da izrazim zahvalnost svom mentoru, prof. dr Nikoli Žariću, čija su stručnost, podrška i smjernice imale značajnu ulogu u odabiru teme i uspješnom usmjeravanju mog istraživačkog rada.

APSTRAKT

U ovom radu istražuje se primjena vještačke inteligencije u automatizovanoj klasifikaciji i analizi mail-a, s ciljem pouzdane detekcije phishing mail-ova u realnim okruženjima privatne i univerzitetske komunikacije. Kao rezultat istraživanja razvijena je Streamlit aplikacija koja omogućava koja omogućava praktičnu primjenu razvijenih modela.

Korišćeni su javno dostupni skupovi podataka Spam-Assassin, Enron, CEAS 08, TREC-05, TREC-06 i TREC-07, koji su nakon čišćenja, balansiranja SMOTE tehnikom i standardizacije formirani u jedinstveni korpus sa jasno označenim klasama. Primijenjeni su klasični algoritmi Naive Bayes i Support Vector Machine kao i napredni NLP modeli DistilBERT, LoRa i Mistral LLM, uz tehnike TF-IDF vektorizacije i fine-tuning prilagođavanja.

Rezultati su pokazali da kombinovanje klasičnih modela i modela dubokog učenja daje izuzetno dobre performanse. Klasični pristupi poput NB i SVC ostvarili su tačnost od 95.64 % i 96.52 %, dok su napredni NLP modeli značajno poboljšali rezultate. DistilBERT je dostigao 98.5 %, LoRa model 97.64 %, a full fine-tuning verzija 98.16 % tačnosti. Poseban iskorak napravio je Mistral LLM, koji je doprinio dubljem razumijevanju i objašnjavanju sadržaja mail-ova, omogućavajući semantičku interpretaciju konteksta i pouzdaniju detekciju sofisticiranih phishing pokušaja.

Zaključeno je da sinergija klasičnih algoritama i modela dubokog učenja predstavlja visoku tačnost, robusnost i interpretabilnost, kombinujući efikasnost tradicionalnih metoda sa interpretativnom snagom velikih jezičkih modela. Ovim radom potvrđeno je da hibridni sistemi bazirani na vještačkoj inteligenciji mogu značajno unaprijediti sigurnost digitalne komunikacije i pružiti temelje za razvoj naprednih alata za zaštitu korisnika.

Ključne riječi: vještačka inteligencija, phishing detekcija, klasifikacija mail-ova, NLP, Mistral LLM

ABSTRACT

This paper explores the application of artificial intelligence in the automated classification and analysis of emails, with the goal of achieving reliable detection of phishing messages in real-world private and university communication environments. As a result of the research, a Streamlit application was developed to enable the practical implementation of the proposed models.

Publicly available datasets — Spam-Assassin, Enron, CEAS 08, TREC-05, TREC-06, and TREC-07 — were used, which were cleaned, balanced using the SMOTE technique, and standardized into a unified corpus with clearly labeled classes. Classical algorithms such as Naive Bayes (NB) and Support Vector Machine (SVM) were applied, along with advanced NLP models DistilBERT, LoRa, and Mistral LLM, using TF-IDF vectorization and fine-tuning techniques for adaptation.

The results demonstrate that combining classical machine learning models with deep learning models yields excellent performance. Classical approaches such as NB and SVC achieved accuracy rates of 95.64% and 96.52%, respectively, while advanced NLP models significantly improved the results — DistilBERT reached 98.5%, the LoRa model 97.64%, and the full fine-tuning version 98.16% accuracy. A notable advancement was achieved with Mistral LLM, which contributed to deeper understanding and explanation of email content, enabling semantic interpretation of context and more reliable detection of sophisticated phishing attempts.

It was concluded that the synergy between classical algorithms and deep learning models provides high accuracy, robustness, and interpretability, combining the efficiency of traditional methods with the interpretive power of large language models. This study confirms that hybrid AI-based systems can significantly enhance the security of digital communication and provide a foundation for the development of advanced tools for user protection.

Keywords: artificial intelligence, natural language processing (NLP), email classification, phishing detection, Mistral LLM

Sadržaj

1. Uvod

Mail ostaje jedan od najrasprostranjenijih oblika komunikacije, kako u poslovnom tako i u privatnom okruženju. Nažalost, mail je i primarni vektor za phishing napade u kojima napadači putem lažnih poruka pokušavaju obmanuti korisnike da odaju povjerljive informacije ili izvrše određene radnje na svoju štetu. Takve zlonamjerne mail poruke često imitiraju legitimne izvore, čime vješto navode primaoce da teško uoče prevaru. Cilj phishing-a je najčešće krađa osjetljivih podataka poput lozinki, brojeva kreditnih kartica ili finansijskih informacija, koje napadači potom zloupotrebljavaju za neovlašćen pristup računima, krađu identiteta i finansijsku dobit. Zbog ogromne količine mail saobraćaja i sve veće učestalosti phishing napada, automatizovana sigurnosna rješenja postala su neophodna. Primjera radi, samo u periodu od novembra 2023. do januara 2024. zabilježeno je oko milion prijava phishing incidenata, što ilustruje razmjere ovog problema. Istovremeno, tehnike phishing-a stalno evoluiraju od jednostavnih lažnih poruka prevaranti su prešli na vrlo uvjerljive socijalno-inženjerske šeme i višekanalne napade koje klasični filteri sve teže prepoznaju. Ovakav razvoj situacije nameće potrebu za naprednijim pristupima zaštiti mail komunikacije, uključujući primjenu savremenih metoda vještačke inteligencije za detekciju i analizu sumnjivih poruka. Rani pokušaji zaštite mail-a oslanjali su se na ručna pravila i crne liste za filtriranje nepoželjnih poruka. Kasnije su uvedene metode mašinskog učenja kao što je Naivni Bayesov (NB) klasifikator i mašine sa potpornim vektorima (SVM) koje su postale standardni alati za filtriranje spam-a i phishing mail-ova. Ovi klasični modeli koriste inženjering karakteristike da bi razlikovali legitimne i phishing mail-ove. NB i SVM pokazali su se relativno uspješnim, a studije su izvijestile o tačnostima detekcije phishing mail-ova i preko *90%* za ove algoritme, pri čemu SVM nerijetko blago nadmašuje NB (Abu-Nimeh et al. 2007). Na primjer, Unnithan i saradnici postigli su tačnost od 94,3% korištenjem SVM-a (uz 93,4% za Naivni Bayes) na skupu podataka phishing mail-ova. Međutim, uprkos solidnim rezultatima, ograničenje klasičnih modela leži u njihovoj zavisnosti od ručno izabranih karakteristika i ograničenoj sposobnosti da razumiju kompleksniji kontekst poruke. Kako su napadi postajali sofisticiraniji, istraživači su počeli primjenjivati složenije modele mašinskog učenja (Unnithan et al. 2018).

Tokom protekle decenije, duboko učenje donijelo je značajan napredak u zadatku klasifikacije mail-ova. Umjesto ručnog definisanja karakteristika, duboki neuronski mrežni modeli mogu automatski naučiti relevantne osobine teksta mail-a i time poboljšati tačnost detekcije phishing-a. Brojne studije pokazale su da duboki modeli nadmašuju klasične pristupe u prepoznavanju zlonamjernih poruka. Modeli dubokog učenja mogu efikasnije prepoznati

suptilne obrasce u mail tekstu koje statički filtri ili jednostavnije metode možda propuštaju (Sahingoz et al. 2019). Istraživanja su se proširila na različite neuronske arhitekture za analizu mail sadržaja. Duboko učenje je danas temelj mnogih naprednih sistema za filtriranje phishing-a, ali zahtijeva značajne računarske resurse i obilje podataka za obuku.

Najnovija faza u razvoju alata zasnovanih na vještačkoj inteligenciji (AI) za analizu teksta uključuje velike jezičke modele poput Bidirectional Encoder Representations from Transformers (BERT). Ovi modeli, trenirani na ogromnim korpusima teksta, pokazuju izuzetnu sposobnost razumijevanja konteksta i generisanja jezika bliskog ljudskom. U domenu mail sigurnosti, veliki jezički modeli (LLM) se prirodno nameću kao moćan alat i mogu da analiziraju semantičko značenje poruke, prepoznaju suptilne kontekstualne znakove phishing-a i čak generišu objašnjenja ili sažetke sadržaja poruke. Istraživanja su već otpočela primjenu LLM modela u detekciji phishing-a. Na primjer, zabilježeno je da su kombinacije transformera sa najnovijim GPT-4 modelom postigle skoro 99% tačnosti na standardnim skupovima podataka (Smith et al. 2024). Najbolji LLM-ovi broje desetine milijardi parametara i zahtijevaju ogromnu računarsku snagu. Zbog toga su aktuelna istraživanja usmjerena i na manje, optimizovane LLM-ove, koji bi uz prihvatljivije resurse pružili slične benefite. Značajan primjer predstavlja Mistral 7B, otvoreni jezički model sa sedam milijardi parametara, predstavljen 2023. godine, koji u više standardnih zadataka obrade prirodnog jezika (Natural Language Processing, NLP) dostiže ili čak nadmašuje performanse znatno većih modela, uz znatno manju potrošnju memorije (Aljofey et al. 2022). Takvi modeli najavljuju novu eru dostupnosti napredne AI analitike teksta. Ipak, dosadašnje studije ukazuju da sami LLM-ovi bez prilagođavanja još nisu superiorni u odnosu na klasične ili duboke modele za detekciju phishing-a po čistoj tačnosti detekcije. Na primjer, Thapa i saradnici izvještavaju da su njihovi kvantizovani LLM-ovi manjeg obima ostvarili nešto nižu tačnost od klasičnih algoritama mašinskog učenja (ML) na istom zadatku phishing detekcije (Thapa et al. 2025). S druge strane, LLM pristup je pokazao prednost u prepoznavanju suptilnih, kontekstualnih znakova phishing-a koje tradicionalni modeli mogu propustiti, kao u pružanju dodatne interpretabilnosti i generisanju kratkih objašnjenja uz svoju odluku (Thapa et al. 2025). Ovi uvidi sugerišu da postoji prostor da se kombinuju klasični modeli i LLM i na taj način dolazi do spajanja visoke preciznosti specijalizovanih klasifikatora sa kontekstualnom “inteligencijom” velikih jezičkih modela.

Složenost modernih phishing napada implicira da nijedna pojedinačna metoda ne obezbjeđuje potpunu zaštitu. Klasični mašinski algoritmi pružaju robustnost i efikasnost, brzi su, zahtijevaju manje resursa i dobro rade na uobičajenim obrascima koje su naučili. Duboki

modeli donose veću preciznost na račun složenosti, dok LLM-ovi dodaju kontekstualno razumijevanje i mogućnost analize sadržaja izvan puke klasifikacije. Kombinovanjem ovih pristupa nastoji se iskoristiti najbolje od oba svijeta. Literatura sve više naglašava prednosti hibridnih rješenja, integracija LLM-ova sa provjerenim metodama klasičnog ML može poboljšati otpornost sistema na nove, neviđene oblike napada. Na primjer, LLM se može iskoristiti da prepozna kontekstualne anomalije ili adaptivne phishing taktike, dok paralelno tradicionalni klasifikator pouzdano hvata već poznate obrasce. Višestruki modeli se mogu kombinovati u ansambl koji donosi robusniju odluku od bilo kojeg pojedinačnog modela. Zapaženo je i da LLM-ovi mogu poslužiti za generisanje objašnjenja uz odluku klasifikatora, što povećava povjerenje korisnika u automatizovani sistem i omogućava dublju analizu sadržaja mail-a, a ne samo binarnu odluku. Svi ovi razlozi motivišu istraživanje kombinovanog pristupa upravo u ovom radu.

U okviru ovog master rada razvija se i evaluira automatizovani sistem za klasifikaciju i analizu mail poruka primjenom savremenih metoda AI. Cilj ovog istraživanja jeste unapređenje detekcije phishing mail poruka kroz integraciju tradicionalnih metoda mašinskog učenja, savremenih modela dubokog učenja i LLM-a. Problem detekcije phishing-a, iako istraživan već duži niz godina, ostaje izazov zbog sve sofisticiranijih tehnika napada i varijabilnosti tekstualnog sadržaja. Zbog toga se u ovom radu pristupa razvoju hibridnog sistema koji objedinjuje više tehnoloških paradigmi u cilju postizanja bolje tačnosti, objašnjivosti i otpornosti na nove prijetnje.

U prvoj fazi rada, fokus je stavljen na analizu postojećih pristupa u klasifikaciji mail poruka, obuhvatajući metode kao što su NB i SVM, koje se svrstavaju u klasične algoritme, zatim neuronske mreže bazirane na dubokom učenju, kao i najnovije pristupe temeljene na velikim jezičkim modelima poput Mistral AI.

Implementiran je eksperimentalni sistem koji kombinuje klasične klasifikatore sa LLM pristupom u jedan integrisani hibridni model. Ovaj sistem koristi javno dostupne skupove podataka za obuku i testiranje, što omogućava ponovljivost eksperimenta i uporedivost rezultata sa postojećim rješenjima. Takvi skupovi obuhvataju legitimne i phishing poruke pa predstavljaju realističan izazov za sve primijenjene modele.

Nakon implementacije, sistem je podvrgnut opsežnoj evaluaciji koja uključuje standardne metrike kao što su tačnost, preciznost, odziv (recall) i F1 skor (Chawla et al. 2002). Kroz poređenje performansi različitih modela pojedinačno i u kombinaciji procijenjena je vrijednost hibridnog pristupa. Posebna pažnja posvećena je tome kako modeli reaguju na do

sada neviđene phishing obrasce, kao i koliko dobro balansiraju između efikasnosti i objašnjivosti klasifikacionih odluka.

Konačno, razvijen je i aplikativni sloj sistema kroz korišćenje Streamlit frameworka, čime je omogućena interaktivna demonstracija funkcionalnosti modela. Aplikacija omogućava korisnicima da izaberu mail sa privatnog ili univerzitetskog naloga, nakon čega sistem automatski vrši klasifikaciju i prikazuje rezultate analize. Osim same binarne klasifikacije korisniku je omogućeno da postavi dodatna pitanja AI modelu radi boljeg razumijevanja odluke. Time je dodatna vrijednost rada usmjerena ka stvaranju alata koji ne samo da prepoznaje prijetnje, već i edukuje korisnika kroz objašnjavanje uzroka odluka i potencijalnih bezbjednosnih rizika.

Ovaj rad nastoji dati doprinos unapređenju sigurnosti mail komunikacije kroz sinergiju provjerenih algoritama mašinskog učenja i najnovijih AI modela. Korištenjem kombinovanih metoda na javno dostupnim podacima i kroz razvoj demonstracione aplikacije, pokazaćemo kako se moderni koncepti vještačke inteligencije mogu iskoristiti za pouzdaniju detekciju phishing mail-ova i detaljniju analizu sadržaja poruka. Realizacijom istraživanja predviđa se razvoj inteligentnog, prilagodljivog i transparentnog sistema za detekciju phishing mail-ova koji kombinuje tradicionalne algoritme mašinskog učenja i moderne modele dubokog učenja.

Cilj je da se pokaže kako integracija više pristupa može unaprijediti preciznost klasifikacije, smanjiti broj lažno pozitivnih i lažno negativnih predikcija te omogućiti bolje razumijevanje odluka modela. Poseban doprinos rada ogleda se u razvoju interaktivne aplikacije koja korisnicima omogućava da u realnom vremenu testiraju sistem, analiziraju rezultate i dobiju objašnjenja za klasifikacione odluke. Na taj način se ne postiže samo povećanje tačnosti, već i jačanje povjerenja korisnika u AI sisteme kroz njihovu transparentnost i interpretabilnost.

Istraživanje koristi kombinovane metode iz oblasti mašinskog učenja, analize podataka, automatizacije i interpretabilnosti modela. Za klasifikaciju phishing mail-ova primjenjuju se klasični algoritmi NB i SVM, kao i modeli dubokog učenja DistilBERT i Mistral. Efikasnost modela biće procijenjena standardnim metrikama kao što su preciznost, odziv, F1-score i ROC-AUC. Obrada teksta uključuje uklanjanje HTML oznaka, tokenizaciju, normalizaciju, lematizaciju i prepoznavanje ključnih riječi koje se često javljaju u phishing porukama. Za balansiranje skupa podataka koristiće se SMOTE metoda, dok će VirusTotal API služiti za reputacione provjere domena i IP adresa pošiljalaca. Dodatno, podaci će se prikupljati

korišćenjem Selenium WebDriver-a za univerzitetske naloge i Gmail API-ja za privatne mail-ove, čime se osigurava realna i raznovrsna baza za eksperiment.

Poseban segment istraživanja odnosi se na explainable AI pristupe koji omogućavaju interpretaciju odluka modela. Za vizuelnu interpretaciju rezultata korišćene su matrice konfuzije, ROC krive i WordCloud dijagrami, čime je omogućeno grafičko poređenje performansi modela i analiza tekstualnih karakteristika phishing poruka. Korisnik na ovaj način može razumjeti zbog čega je određeni mail označen kao phishing, što doprinosi transparentnosti i prihvatanju sistema. Procjenjuje se da će ovakav kombinovani pristup potvrditi hipoteze rada, pokaže mjerljivo unapređenje u performansama modela i doprinese razvoju praktično primjenjivog alata za sigurniju elektronsku komunikaciju.

2. Pregled tehnika za detekciju phishing mail-ova

Detekcija phishing poruka predstavlja jedno od ključnih istraživačkih polja u oblasti sajber bezbjednosti. Tokom proteklih decenija razvijen je veliki broj metoda koje se mogu svrstati u tri glavne grupe: klasični algoritmi mašinskog učenja, savremeni pristupi zasnovani na dubokom učenju i LLM-u. Svaka od ovih grupa ima svoje prednosti i ograničenja, a njihova primjena zavisi od dostupnih resursa, kvaliteta podataka i ciljeva sistema. U nastavku slijedi detaljan pregled ovih tehnika uz naglasak na njihovu evoluciju i upotrebu u savremenim istraživanjima.

Jedan od najranijih i najjednostavnijih pristupa detekciji phishing mail-ova bio je oslanjanje na NB klasifikator. NB pretpostavlja uslovnu nezavisnost atributa i računa vjerovatnoću da poruka pripada klasi phishing ili legitimnih mail-ova na osnovu frekvencije riječi i drugih karakteristika. Ova tehnika se pokazala vrlo efikasnom krajem 1990-ih za filtriranje neželjene pošte, a prednost Naivnog Bayesa je brzina i mali računarski zahtjevi. On može brzo obraditi velike količine mail-ova u realnom vremenu. U eksperimentima je često ostvarivao visoku tačnost na uravnoteženim skupovima podataka. Na primjer, Yasin i Abuhasan koriste NB kao dio svog modela i postižu preko 95% tačnosti na eksperimentalnom skupu (Yasin and Abuhasan 2016). Međutim, NB ima svoja ograničenja, a njegov učinak opada na kompleksnijim skupovima podataka gdje su obilježja korelirana. Ujedno je sklon proizvodnji većeg broja lažno pozitivnih/negativnih označavanja ako se ne primijene odgovarajuće tehnike podešavanja.

SVM algoritmi su postali popularni za detekciju phishing i spam poruka krajem 1990-ih i početkom 2000-ih. Drucker i saradnici su među prvima pokazali da SVM može nadmašiti tradicionalne metode u filtriranju mail-ova. Osnovna ideja SVM-a je pronalaženje optimalne hiper-ravni koja razdvaja klase uz maksimalnu margin (Drucker et al. 1999). U praksi se mail poruke predstavljaju kao visokodimenzionalni vektori, a SVM se vrlo dobro nosi s takvom dimenzionalnošću i ima izvrsnu sposobnost generalizacije (Yasin and Abuhasan 2016). Istraživanja su pokazala da SVM sa linearnim jezgrom na velikim skupovima podataka pruža visoku tačnost uz relativno brzu klasifikaciju. Abu-Nimeh i saradnici uporedili su više klasifikatora za detekciju phishing-a i utvrdili da je SVM bio među najtačnijima. Njihovi eksperimenti su koristili preko 7000 atributa i pokazali da SVM posebno dobro radi uz binarizaciju karakteristika (Abu-Nimeh et al. 2007). Kao i kod NB, glavni nedostatak SVM-a je složenost treniranja na zaista velikim skupovima podataka. Algoritamska kompleksnost

može biti problem kada broj mail-ova mjeri stotinama hiljada ili više ali napredak u optimizaciji SVM-a omogućio je efikasnije treniranje linearnih SVM-ova na velikim skupovima podataka. SVM modeli su korišćeni ne samo za analizu tekstualnog sadržaja mail-a, već i za analizu URL adresa sadržanih u mail-u. SVM se afirmisao kao moćan alat za ranu detekciju phishing-a, mada uz cijenu kompleksnijeg treniranja. Klasični algoritmi poput NB i SVM dali su temeljne rezultate, brzi su i relativno jednostavni za implementaciju, ali im nedostaje fleksibilnost dubljeg razumijevanja sadržaja mail-a kakvo pružaju moderne metode zasnovane na učenju reprezentacija.

Revoluciju u NLP-u, uključujući detekciju phishing-a, donijela je Transformer arhitektura i unaprijed obučeni jezički modeli zasnovani na njoj. BERT model označio je prekretnicu (Devlin et al. 2019). To je dvosmjerni transformer pretreniran na ogromnom korpusu teksta, koji pruža kontekstualizovane reprezentacije riječi. BERT omogućava razumijevanje suptilnih nijansi u mail-u: npr. model može razumjeti da je rečenica "Dear user, your account is suspended" vjerovatno phishing, dok "Dear user, your account *has been updated* as requested" nije, iako su riječi slične, a razlika je u semantičkom kontekstu i tonu, što BERT uspješno prepoznaje. Mnogi radovi su brzo primijenili BERT za klasifikaciju mail-ova. Modeli zasnovani na BERT-u dali su odlične rezultate u detekciji, pri čemu je DistilBERT nudio najbolji kompromis brzine i tačnosti (Songailaitė et al. 2023). DistilBERT je posebno zanimljiv kada je riječ o *distilovanom* modelu koji je 40% manji od originalnog BERT-a, zadržavajući oko 97% njegove performanse, uz 60% brže izvršavanje. To ga čini pogodnim za operativnu upotrebu u sistemima gdje se svaka poruka mora brzo analizirati. Još jedna prednost BERT-modela je mogućnost da se dodatno obučava (fine-tune) za konkretan skup podataka. Dovoljno je dodati jednostavan klasifikacioni sloj na vrh i dalje trenirati model nekoliko epoha na primjerima phishing mail-ova. Duboki pristupi zasnovani na neuronskim mrežama značajno su unaprijedili mogućnosti detekcije phishing-a. Oni automatski uče reprezentacije iz podataka, zaobilazeći potrebu za ručnim inženjeringom obilježja. Naravno, cijena je veća potreba za podacima i resursima pa ovi modeli obično zahtijevaju stotine hiljada označenih mail-ova za trening, kao i upotrebu snažnih grafičkih procesora (GPU) za efikasno treniranje. Jedna često primjenjivana tehnika u radu sa phishing datasetovima je balansiranje klasa. Budući da legitimni mail-ovi obično daleko brojčano nadmašuju phishing primjere, koristi se oversampling manjinske klase ili algoritmi poput Synthetic Minority Oversampling Technique (SMOTE), kako bi se izbjegla pristrasnost modela ka većinskoj klasi. Primjena SMOTE-a ili njegovih varijanti u kombinaciji sa dubokim modelima pokazala je poboljšanje recall-a za phishing klasu u neuravnoteženim skupovima podataka.

Najnoviji trend u detekciji phishing-a je primjena LLM-a. Neuralne mreže su trenirane na ogromnim količinama tekstualnih podataka, koje mogu da generišu i razumiju tekst na nivou bliskom ljudskom. Primjeri uključuju OpenAI GPT seriju (GPT-3, GPT-4), Meta-ine modele *LLaMA 1 i 2*, te novije modele kao što je *Mistral 7B*. LLM-ovi sadrže opšte jezičko razumijevanje i mogu se primijeniti kao efikasni klasifikatori phishing poruka putem ciljano formulisanih upita. Kao generativni asistenti, oni dodatno omogućuju objašnjenja i dvosmjernu interakciju s korisnikom. Istraživanja su pokazala da dovoljno veliki jezički modeli mogu postići impresivne rezultate i u detekciji prevara bez eksplicitnog treniranja za tu namjenu. Predstavljen je poboljšani LLM-zasnovan pristup za klasifikaciju mail-ova, koji koristi predtreniranost modela i kombinuje DistilBERT i RoBERTa (Jamal and Wimmer 2023). Prednost LLM-a je što već zna jezik i on je tokom pretreniranja možda već vidio stotine hiljada phishing primjera i naučio. *Meta-in LLaMA* model demonstrirao je da otvoreni modeli sa 7–65 milijardi parametara, trenirani samo na javnim podacima, mogu doseći ili nadmašiti učinak zatvorenih modela (Touvron et al. 2023). Mistral AI tim je 2023. objavio model Mistral 7B koji postiže performanse uporedive s mnogo većim modelima. Ova tehnološka poboljšanja omogućavaju da se danas razviju i interno hostuju relativno lagani modeli, koji uprkos maloj veličini posjeduju visok kapacitet razumijevanja teksta. Za detekciju phishing-a, takvi modeli mogu biti fino podešeni na korpus phishing mail-ova. Istražuje se čak i federativno učenje sa LLM-ovima za phishing detekciju, scenarije gdje se model trenira decentralizovano na mail podacima više organizacija, što je važno zbog privatnosti (Thapa et al. 2023). Veliki jezički modeli unose i nove dimenzije jer mogu objasniti zašto je neka mail poruka potencijalno phishing, što je danas jako važno za korisnike i administratore. Ljudska greška predstavlja i dalje glavni faktor kompromitacije bez obzira na sve metode i filitere, korisnik ipak klikne na link. Mistral 7B je sposoban da naglasi ključne indikatore prevara i da sugeriše šta dalje treba da se radi. Uddin i Sarker demonstriraju upotrebu DistilBERT modela *uz LIME i Transformer Interpret XAI* alate za objašnjenje odluka modela u detekciji phishing-a. Na ovaj način se došlo i do vizualizacije riječi u mail-u koje su najviše doprinijeli odluci phishing, što je feedback za analitičare (Uddin and Sarker 2024). Iako sve zvuči idealno, veliki modeli imaju i određene izazove. Oni su računarski zahtjevni, potrebno je dosta memorije i CPU/GPU snage. Zato se često predlaže hibridni pristup. Koriste se klasični ili duboki modeli za brzu klasifikaciju, a LLM se koristi samo kada korisnik traži objašnjenje. Drugi veliki problem je što LLM mogu generisati tekst koji zvuči uvjerljivo ali je netačan. S obzirom na to sama implementacija mora biti praćena oprezom, a potrebno je i ograničiti njihovo generisanje na povjerljive izjave i idealno kombinovati sa rule-based provjerama. Takođe, danas napadač pokušava da ubaci zlonamjerna upustva u tekst mail-a kako bi pokušao zbuniti model ili ga naveo da propusti

prijetnju. Uprkos ovim izazovima, LLM predstavlja uzbudljiv dodatak odbrani protiv phishing-a. Oni spajaju detekciju i odgovor**,** ne samo da filtriraju prijetnje, već mogu i komunicirati sa korisnicima o tim prijetnjama.

Prednost klasičnih modela je što su veoma brzi, resursno efikasni i jednostavni za implementaciju. Mogu se trenirati i izvoditi na običnim računarima, pa su pogodni za ugrađene sisteme ili masivni streaming, a jedna od mana im je što oslanjaju na ručno dizajnirane karakteristike. Ako bitan indikator phishing-a nije unaprijed osmišljen i izračunat kao karakteristika, model ga neće sam naučiti. Imaju ograničenu sposobnost da shvate kontekst pa tako NB gleda riječi nezavisno, SVM efektivno radi linearno odvajanje u prostoru karakteristika, pa im može promaknuti semantička značenja rečenica. U scenarijima gdje napadači mijenjaju taktiku klasični modeli mogu zahtijevati ručno ažuriranje karakteristika ili retrening. Ipak, i danas se ovi algoritmi koriste kao osnvona referenca i polazna tačka zbog svoje brzine i jednostavnosti. Savremeni pristupi mogu značajno povećati tačnosti i recall detekcije jer automatski uče iz podataka. Takođe mogu i da uoče suptilne obrase koje čovjek ne bi lako definisao ručno. Duboki modeli mogu naučiti da stil pisanja u phishing mail-u drugačiji od legitimnog poslovnog dopisa. Mana im je što su spori u odnosu na klasične algoritme i zahtjevaju snažan hardver. Još jedna mana može biti manjak interpretabilnosti jer duboke mreže su notorno crne kutije. Bez dodatnih alata, teško je objasniti zašto je model odlučio da je mail phishing. To može biti problem u korporativnim okruženjima gdje administratori žele znati razlog alarma.

Potpuni fine-tuning podrazumijeva dalje treniranje kompletnog skupa parametara prethodno pretreniranog jezičkog modela na specifičnom skupu podataka. Suština ove metode je da se model nauči da prilagodi sve svoje parametre kako bi što bolje obavljao novi zadatak. Prednost ovog pristupa je što model ima maksimalnu slobodu da se prilagodi novom domenu tako potencijalno ostvarujući visoku tačnost. Potpuno fino podešavanje velikih modela dolazi uz značajne troškove. Potpuno fino podešavanje nosi rizik zaboravljanja, pa se model može previše optimizovati na novom domenu i pritom narušiti ranije naučene opšte jezičke sposobnosti. U kontekstu detekcije phishing mail-ova, tradicionalni pristup je dugo bio fino podešavanje manjih pretreniranih modela na kolekcije phishing poruka. Takav pristup može dati dobre rezultate, ali za najsavremenije LLM-ove sa stotinama miliona ili milijardama parametara, potpuni fine-tuning predstavlja ozbiljan izazov u pogledu memorije, vremena obuke i skalabilnosti.

Pored potpunog fine tuninga alternativni pristup predstavlja LoRa koja omogućava efikasnije prilagođavanje velikih modela novim zadacima uz znatno manji broj treniranih parametara. Metodu su uveli Hu i saradnici sa ciljem rješavanja problema standardnog fine-tuninga na LLM-ovima. Ključna ideja LoRa je da se težine originalnog modela ostanu nepromijenjene, a da se prilagođavanje ostvari dodavanjem malih tenzora niskog ranga u svakom sloju mreže koji se treniraju za dati zadatak. Za odabrane velike matrice težina u modelu LoRa uvodi dodatne matrice znatno manjih dimenzija čiji se proizvod dodaje izvornim težinama. Prilikom treniranju samo te male matrice se optimizuju, dok originalne težine ostaju fiksne. Rezultat je zapravo efikasan ažurirani skup težina modela nižeg ranga. Ovakav pristup dramatično smanjuje broj slobodnih parametara koje je potrebno naučiti (Hu et al. 2021). U mnogim slučajevima, LoRa omogućava da se fino podešavanje obavi podešavanjem manje od 1% ukupnih parametara modela. Praktična posljedica je ogromna ušteda memorije i računarskog vremena dje umjesto da optimizujemo milijarde parametara, model uči samo mali dodatak. Hu i saradnici su pokazali da LoRa može smanjiti broj treniranih parametara i do 10.000 puta uz trostruko manju potrebu za GPU memorijom (Hu et al. 2021). Istovremeno, performanse tako prilagođenog modela ostaju uporedive sa potpunim fine-tuning pristupom. Još jedna prednost LoRa pristupa je očuvanje osnovnih znanja modela. Pošto se bazne težine ne mijenjaju, originalno pretrenirano znanje ostaje netaknuto ispod sloja prilagođavanja. Model se može vraćati na originalno stanje jednostavnim uklanjanjem LoRa modula ili se različiti LoRa moduli mogu razmjenjivati na isti bazni model za različite zadatke. Sve ove karakteristike čine LoRa izuzetno privlačnim za prilagođavanje LLM-ova u scenarijima gdje su resursi ograničeni ili je poželjno zadržati jedinstveni bazni model za više namjena. Obje opisane metode imaju svoje uloge, prednosti i nedostatke. Potpuni fine-tuning podešava 100% parametara modela pa se praktično cijela mreža nanovo trenira na ciljnom zadatku (Hu et al. 2021). Nasuprot tome LoRa mijenja tek mali procenat težina dodajući nizak rang prilagodbe uz očuvanje većine originalnih parametara. Manji broj treniranih parametara znači i manji rizik od prenaučenosti na mali skup podataka, što je korisno kada je dostupno ograničeno primjeraka phishing mail-ova za treniranje. Potpuni fine-tuning teoretski pruža maksimalnu fleksibilnost modelu da se prilagodi, što može dovesti do vrhunskih performansi ako su zadovoljeni uslovi. Uz pažljivo podešavanje ranga i hiperparametara, LoRa ne žrtvuje tačnost u zamjenu za efikasnost. Ipak, u ekstremnim slučajevima dje novi zadatak zahtijeva značajnu reorganizaciju znanja modela, potpuni fine-tuning bi mogao dostići nešto bolji rezultat jer ne ograničava promjene na niskorangiranu aproksimaciju.

3. Analiza i obrada podataka

Analiza i obrada podataka predstavljaju ključnu fazu u razvoju sistema za automatsku detekciju phishing mail-ova. S obzirom na raznovrsnost sadržaja mail-ova kao i na njihovu složenost bilo je neophodno izvršiti prečišćavanje sirovih tekstualnih podataka, ekstrakciju relevantnih osobina kao i pripremu podataka za obuku različitih modela mašinskog učenja. Cilj ovog poglavlja je da prikaže sve korake od izbora dataset-ova, normalizacije teksta, analize strukture mail-a, tokenizacije i uklanjanja riječi bez semantičkog značenja, vektorizacije teksta pomoću obrnute učestalosti dokumenata (Term Frequency – Inverse Document Frequency, TF-IDF), konstrukciju ulaznih osobina za modele kao i promjena heurističkih bezbjednosnih pravila. Akcenat je dat na evaluaciji kvaliteta podataka, balansu klasa kao i kombinovanju modela zbog postizanja veće tačnosti i otpornosti sistema. U cilju rješavanja problema neravnoteže podataka, korišćena je tehnika sintetičkog naduzorkovanja poznata kao SMOTE, kojom se balansiraju klase kreiranjem novih sintetičkih uzoraka manjinske klase (Chawla et al. 2002). Mail sadrži niz informacija raspoređenih u različite strukuralne cjeline, zaglavlje, naslov, glavno tijelo poruke, kao i prilog i URL linkove. Obrada podataka je morala obuhvatiti i sigurnosne signale kao što su sumnjivi linkovi, fajlovi sa zlonamjernim ekstenzijama i IP adrese pošiljaoca koje su već identifikovane kao maliciozne. Korišteni su javno dostupni datasetovi koji sadrže hiljade mail-ova koji su označeni kao legitimne ili phishing poruke. Ovi skupovi su očišćeni i konvertovani u zajednički tekstualni format kako bi se omogućila kvalitetna obrada. Tekstualni podaci su prošli kroz nekoliko faza predobrade, uklanjanje HTML oznake, tokenizacija, normalizacija u mala slova, filtiriranje zaustavnih riječi kao i lematizacija i svođenje riječi na korijenski oblik. Svođenje riječi na korijen predstavlja process redukcije riječi na njihov osnovni morfološki oblik uklanjanjem sufiksa i prefiksa. Prečišćeni podaci su korišćeni za generisanje numeričkih reprezentacija uz pomoć TF IDF vektora i transformera baziranih na BERT arhitekturi. TF-IDF vektorizacija omogućila je ocjenjivanje značajnosti svake riječi u odnosu na cijelokupni korpus i pružila je jednostavnu, ali efektivnu reprezentaciju teksta za klasične algoritme mašinskog učenja (Ramos 2003). Parelelno su korištene i heurističke metode kako bi se prepoznali sumnjivi domeni, skraćeni URL-ovi i ekstenzija fajlova. Svi pripremljeni podaci iskorišćeni su za treniranje više modela klasifikacije kao što su NB, Linear SVC i DistilBERT. Pored ovih pristupa, testiran je i Mistral AI, savremeni jezički model optimizovan za visoku efikasnost i rad sa složenim kontekstima, što ga čini pogodnim za detekciju suptilnih obrazaca u podacima. Kod DistilBERT modela, korišten je ubrzani fine-tuning na uzorku od 20% podataka kako bi se brzo dobila početna evaluacija performansi i identifikovali potencijalni problemi u treniranju. Nakon toga je

sproveden potpuni fine-tuning na kompletnom skupu podataka, čime je model dodatno optimizovan i poboljšana njegova sposobnost generalizacije. Modele smo evaluirani pojedinčano, ali je razvijen i objedinjeni model kako bi se povećala tačnost predikcije. Kombinovanjem različitih modela kroz objedinjeni pristup postignuta je bolja robusnost i veća otpornost sistema na greške pojedinačnih klasifikatora (Rokach 2010). U daljem tekstu biće detaljno objašnjen svaki korak uz primjere, kao i analizirani rezultati.

3.1. Izvori podataka koji su korišteni u istraživanju

Datasetovi uključuju Spam-assassin, Enron, CEAS-08, TREC-05, TREC-06 i TREC-07 koje su preuzete iz javno dostupnih baza podataka (Apache Spam-assassin Project, 2006.; Klimt and Yang, 2004; CEAS Conference, 2008; TREC, n.d.). Oni sadrže tekstualna polja subject, body, from i jasno označene klase. Nakon preuzimanja, svi datasetovi su očišćeni od duplikata i praznih vrijednosti, standardizovani na format ‘label’ (0/1), a tekstualna polja objedinjena radi konzistentne obrade. Ovi datasetovi obuhvataju različite periode (2000–2008), različite stilove komunikacije i omogućavaju robustnu evaluaciju modela spam/phishing detekcije (Androutsopoulos et al. 2000).

Zahvaljujući integraciji više dataset-ova obezbjeđena je pouzdanost prilikom testiranja performansi modela, a raznovrsnost podataka je doprinije la smanjenju pristrastnosti modela i poboljšala njegovu primjenjivost u stvarnom okruženju. Svi preuzeti skupovi podataka sačuvani su unutar direktorijuma u kojem se nalazi izvorni kod, u posebnom podfolderu nazvanom data.

Dataset	Broj mail-ova	Broj kolona	Nazivi kolona	Phishing	Legit
Assassin	5,809	7	sender, receiver, date, subject, body, label, urls	1,718	4,091
CEAS-08	39,154	7	sender, receiver, date, subject, body, label, urls	21,842	17,312
Enron	29,767	3	subject, body, label	13,976	15,791
Ling	4,493	7	sender, receiver, date, subject, body, label, urls	3,153	1,340
TREC-05	59,015	7	sender, receiver, date, subject, body, label, urls	23,558	35,457
TREC-06	16,439	7	sender, receiver, date, subject, body, label, urls	6,654	9,785
TREC-07	53,757	7	sender, receiver, date, subject, body, label, urls	29,399	24,358

Tabela 1. Pregled korišćenih datasetova i njihove osnovne karakteristike

Spam-assassin dataset predstavlja jedan od najranijih i najpoznatijih javno dostupnih skupova podataka za istraživanje detekcije spam poruka. On sadrži različite podskupove kao što su *spam*, *easy_ham*, *hard_ham* pa na taj način omogućava analizu poruka bez obzira na njihovu složenost i nivoa prijetnje. Ovaj dataset ima jasno označene primjere i bogate metapodatke pa se često koristi kao standard u procjeni osnovnih filtera i heurističkih pravila.

Dataset je nastao kao dio rada na istoimenom projektu "Spam-assassin", koji je razvijen u cilju filtriranja neželjenih mail-ova korišćenjem niza pravila i statističkih metoda (Apache Spam-assassin Project 2006)[1]. Poruke su preuzete iz stvarog mail-a, što dodatno doprinosi njegovoj vjerodostojnosti i praktičnoj vrijednosti u istraživanjima. Podskupovi omogućavaju testiranje filtera na porukama koje su slične spamu, ali su legitimne, čime se ispituje otpornost sistema na lažno pozitivne klasifikacije. Spam-assassin dataset omogućava razvoj i evaluaciju različitih modela za detekciju spam-a, uključujući tradicionalne metode poput NB klasifikatora, kao i savremene pristupe zasnovane na dubokom učenju.

TREC 2005 skup podataka kreiran je u okviru *TREC Spam Track* takmičenja, koje je organizovano kao dio šire inicijative Teksaškog Nacionalnog Instituta za Standardizaciju i Tehnologiju (NIST) za ocjenu i unapređenje metoda za automatsko filtriranje neželjene pošte (Cormack and Lynam 2005). Skup uključuje preko 90.000 mail-ova, prikupljenih iz različitih izvora i vremenskih perioda, što ga čini jednim od najopsežnijih javno dostupnih dataset-a za ovu oblast. Više od polovine poruka u skupu označeno je kao *spam*, dok je ostatak klasifikovan kao legitimna pošta. Ključna prednost TREC 2005 dataset-a je ujednačeno označavanje poruka pa osim same klasifikacije, dataset sadrži puna zaglavlja i tijela poruka. To omogućava dublju analizu strukture mail poruka, identifikaciju obrazaca ponašanja, kao i meta-informacije i HTML sadržaje.

Dataset je podeljen u više verzija (npr. *spam1*, *spam2*, *ham1*, *ham2* itd.) što nam pruža mogućnost da testiramo otpornost modela na različite tipove podataka i domene. U okviru TREC 2005 definisane su i različite metrike za evaluaciju, uključujući stopu lažno pozitivnih i lažno negativnih klasifikacija, što dodatno olakšava poređenje performansi među modelima.

Nakon uspijeha TREC Spam Track 2005, organizatori su nastavili sa unaprjeđenjem pa skupovi podataka iz 2006. I 2007. predstavljaju prirodni nastavak prethodnog rada.

Datasetovi sadrže desetine hiljada mail poruka koje su prikupljene u stvarnom vremenu sa više korisničkih naloga, a njihova ključna karakteristiha je dinamičnost i vremenska

[1] Apache Spam-assassin Project. (2006). *Spam-assassin public mail corpus*. Apache Software Foundation. https://spam-assassin.apache.org/old/publiccorpus/

komponenta. Oni simuliraju priliv mail-ova kod krajnjih korisnika i time se omogućava istraživanje modela u stvarnom vremenu.

TREC 2006 i 2007 skupovi dodatno uvode pojam "delayed feedback evaluation", gdje evaluacija algoritma zavisi od njegovih ranijih odluka bez mogućnosti naknadnog uvida u podatke iz budućnosti. Kao i prethodni datasetovi, TREC 2006 i 2007 obezbjeđuju puna zaglavlja i tijela poruka, što omogućava analizu širokog spektra atributa, uključujući strukturalne, stilske, sadržajne i tehničke karakteristike mail-ova, a poruke su jasno označene kao *spam* ili *ham*.

CEAS 2008 Live Spam Challenge Corpus predstavlja jedan od najznačajnijih skupova podataka u oblasti filtriranja phishing mail-ova koji je nastao u okviru *Conference on Email and Anti-Spam (CEAS)*, vodeće konferencije posvećene istraživanjima u domenu mail sigurnosti. Za razliku od prethodnih statičkih datasetova, ovaj skup ima za cilj da procjeni efikasnost spam filtera u realnom vremenu. Poruke su označene, a metapodaci o tačnom vremenu prijema omogućavaju napredne analize kao što su detekcija vremenskih obrazaca i reakcija modela na nagli porast spam mail-ova. Njegova ključna vrijednost je što omogućava evaluaciju otpornosti modela na taktike koje se mjenjaju kako bi se zaobišli filteri. CEAS 2008 Live Spam Challenge Corpus predstavlja dragocjen resurs za istraživanja u različitim oblastima kao što su online učenja, real time detekcija spam-a, konceptualne promjene i analize performansi filtera u visokofrekventnim tokovima podataka (Segal, Cormack and Bratko 2008).

Enron Email Corpus jedan je od najpoznatijih i najkorišćenijih skupova podataka u oblasti analize mail-a, posebno kada je riječ o obradi NLP-a, analizi društvenih mreža, otkrivanju prevara, kao i detekciji spam-a i klasifikaciji poruka. Ovaj skup sadrži preko 600.000 mail-ova koje je poslalo i primilo više od 150 zaposlenih u kompaniji, a poruke su strukutuirane po korisničkim nalozima sa jasno definisanim folderima.

Za razliku od specijalizovanih spam dataset-a kao što su Spam-assassin ili TREC, Enron dataset ne sadrži eksplicitno označene spam poruke (Friginal and Hardy 2013).

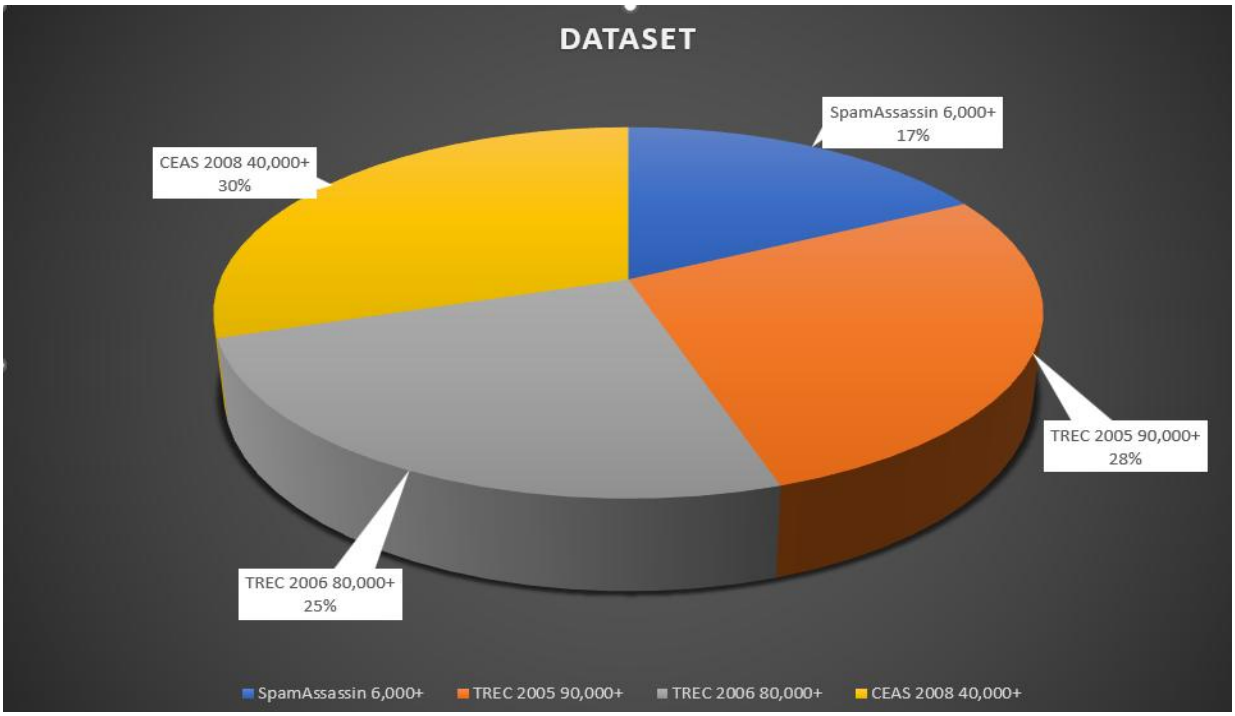
Ovaj korupus je značajan jer omogućava tempolarnu analizu, izgradnju modela komunikacije, klasifikacije mail-ova po temama, prioritetu kao i primjenu tehnike za analizu mreže.

Na osnovu analize predstavljenih dataset-a jasno je da svaki od njih ima specifičnu ulogu u razvoju i evaluaciji sistema za detekciju neželjenih mail-ova. Spam-assassin dataset pruža dobru osnovu za testiranje jednostavnijih heurističkih i statističkih modela zahvaljujući svojoj preglednoj strukturi i klasifikaciji poruka. TREC Spam Corpora (2005–2007) uvode realističniji i kompleksniji eksperimentalni okvir, sa fokusom na vremensku dimenziju i online učenje. CEAS 2008 Live Spam Challenge podiže nivo evaluacije uključivanjem real-time

obrade, simulirajući stvarne tokove mail-ova i adaptivne spam taktike. Enron Email Corpus, iako primarno nije spam dataset, nudi dragocjen uvid u poslovnu komunikaciju i koristi se kao kontrast u treniranju i testiranju sistema za filtriranje.

Zajednički kvalitet ovih dataset-a jeste njihova javna dostupnost i široka upotreba u naučnoj zajednici. Integracijom više skupova podataka moguće je razviti modele koji pokazuju veću robusnost i bolju sposobnost generalizacije.

Svaki skup se razlikuje po broju poruka, procentu spam poruka i prosečnoj dužini poruka. U radu smo koristli Spam-assassin koji sadrži preko 6,000 poruka, od kojih je 35% spam, sa prosečnom dužinom od 120 karaktera. TREC skupove iz 2005. i 2006. Koji imaju znatno veći broj poruka (preko 90,000 i 80,000), veći procenat spam-a (55% i 50%), i duže prosječne poruke (150 i 170 karaktera). CEAS 2008 ima preko 40,000 poruka, sa najvećim procentom spam-a (60%) i prosječnom dužinom od 160 karaktera. Ovi podaci se koriste za treniranje i evaluaciju algoritama za prepoznavanje spam poruka.



Slika 1. Raspodjela korišćenih dataset-ova

3.2. Čišćenje tekstualnog sadržaja, balansiranje podataka uz pomoć SMOTE tehnike, pregled linkova, domena, kao i ključnih riječi u naslovu, vizualizacija sadržaja mail-ova

3.2.1. Čišćenje tekstualnog sadržaja

U okviru detekcije phishing mail-ova izvršena je temeljna obrada sirovih podataka jer tekstualni sadržaj mail-ova često sadrži HTML elemente, neformalne izraze, kodirane sekvence

i različite oblike manipulacije teksta. Kako bi se eliminisao šum u podacima uklonjeni su interpunkcijski i specijalni znakovi korišćenjem regularnih izraza (re.sub), a zatim je tekst podjeljen u riječi i konvertovan u mala slova radi dosljednosti u analizi. U nastavku su primjenjene različite metode za pripremu podataka, balansiranje klasa i ekstrakciju relevantnih karakteristika, a poseban akcenat stavljen je na pregled strukture sadržaja mail-ova kroz analizu naslova, linkova i domena, kao i na vizualizaciju ključnih obrazaca.

Funkcija za dekodiranje Base64URL podataka i ekstrakciju tijela mail poruke

```
  def decode_data(data):
    try:
      missing_padding = len(data) % 4
      if missing_padding:
        data += '=' * (4 - missing_padding)
      return base64.urlsafe_b64decode(data).decode('utf-8', errors='ignore')
    except Exception as e:
      return f"[Greška pri dekodiranju: {e}]"
  body = ""

  if 'body' in payload and 'data' in payload['body']:
    body += decode_data(payload['body']['data'])
  if 'parts' in payload:
    for part in payload['parts']:
      mime_type = part.get('mimeType', '')
      if mime_type == 'text/plain' and 'data' in part.get('body', {}):
        body += decode_data(part['body']['data'])
      elif 'parts' in part:
        body += extract_email_body(part)

return body.strip()
```

Ova funkcija prolazi rekurzivno kroz sve djelove mail-a, a ujedno i praktikuje text/plain MIME tip nad text/html jer tekstualni sadržaj lakše prolazi kroz NLP obradu. Dekodirajuća logika je ključna jer se spriječava mogućnost greške koje često nastaju usljed nekompletnog Base64 sadržaja. Base64 mora imati dužinu djeljivu sa četiri, a u slučaju da nedostaje padding dodaje se znak jednako. Na taj način se omogućava čisto parsiranje tijela poruke. Pored toga,

rekurzivna obrada MIME strukture omogućava izdvajanje sadržaja i kada mail ima više ugnježdenih formata kao što su multipart/alternative i multipart/mixed. Na ovaj način se obezbjeđuje dosljednost u obradi teksta nezavisno od kompleksnosti strukture poruke. Ova strategija je ključna za pripremu kvalitetnog ulaza u NLP pipline jer eliminiše zavisnost od HTML parsiranja i pogrešnog dekodiranja.

```
def classify_email(text, threshold=0.5):
    tfidf = load_model('C:/Users/danil/Desktop/Projekat/data/tfidf_vectorizer.joblib')
    model_nb = load_model('C:/Users/danil/Desktop/Projekat/data/naive_bayes_model.joblib')
    model_svc = load_model('C:/Users/danil/Desktop/Projekat/data/linear_svc_model.joblib')
    text_tfidf = tfidf.transform([text])
    prob_nb = model_nb.predict_proba(text_tfidf)[:, 1]
    prob_svc = model_svc.decision_function(text_tfidf)
    if prob_nb >= threshold or prob_svc >= threshold:
        return "Phishing"
    else:
        return "Legitimate"
```

Početna implementacija imala je za cilj da ukloni znakove interpunkcije i pretvori tekst u mala slova. Phishing mail-ovi postaju složeniji i sadrže HTML elemente, kodirane entitete, skraćene linkove, zamjenske oznake pa je potrebna adekvatna normalizacija. Bez normalizacije ovakvi slučajevi umanjuju kvalitet vektorizacije. Model je radio direktno nad sirovim tekstom što je dovodilo do nepotrebnog šuma usljed HTML oznaka i neuniformne reprezentacije riječi. Zbog ovih mana korišćena je funkcija *normalize_text* uz pomoć koje se smanjio šum u podacima i kreirao stabilniji i upotrebljiviji ulaz za TF-IDF vektorizaciju.

```
    def normalize_text(text):
        """
        Funkcija za normalizaciju tekstualnih podataka.
        Uklanja šum iz teksta kako bi se kreirao stabilniji i upotrebljiviji ulaz
        za TF-IDF vektorizaciju i kasniju klasifikaciju phishing poruka.
```

```
    """
    if not isinstance(text, str):
        return ""
    text = text.lower()
    text = re.sub(r'<.*?>', ' ', text)
    text = re.sub(r'http\S+|www\S+', ' ', text)
    text = text.translate(str.maketrans('', '', string.punctuation))
    text = re.sub(r'\d+', ' ', text)
    text = re.sub(r'\s+', ' ', text).strip()
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in text.split() if word not in stop_words]
    return ' '.join(tokens)
```

Funkcija *normalize_text()* koristi se za čišćenje i standardizaciju tekstualnih podataka prije TF-IDF vektorizacije. Njena svrha je da ukloni nepotrebne elemente iz teksta, kao što su HTML oznake, linkovi, brojevi, interpunkcijski znakovi i često korištene učestale riječi. Takođe, svi karakteri se pretvaraju u mala slova i višestruki razmaci se svode na jedan, čime se dobija ujednačen i čist tekstualni korpus. Ova funkcija se primjenjuje direktno na sadržaj mail poruke prije nego što se izvrši TF-IDF vektorizacija.

```
def classify_email(text, threshold=0.5):
    import numpy as np
    tfidf = load_model('C:/Users/danil/Desktop/Projekat/data/tfidf_vectorizer.joblib')
    model_nb = load_model('C:/Users/danil/Desktop/Projekat/data/naive_bayes_model.joblib')
    model_svc = load_model('C:/Users/danil/Desktop/Projekat/data/linear_svc_model.joblib')
    text_clean = normalize_text(text)
    X = tfidf.transform([text_clean])
    p_nb = float(model_nb.predict_proba(X)[:, 1][0])
    svc_score = float(model_svc.decision_function(X)[0])
```

```
    p_svc = 1.0 / (1.0 + np.exp(-svc_score))
    p_final = 0.5 * p_nb + 0.5 * p_svc
    label = "Phishing" if p_final >= threshold else "Legitimate"
    return label, {"p_final": round(p_final,4), "p_nb": round(p_nb,4), "p_svc": round(p_svc,4)}
```

Uz pomoć ove funkcije postiže se dosljedna reprezentacija što direktno utiče na stabilnost TF-IDF vektora i smanjuje rizik od prenaučenosti na šum. Da bi vjerovatnoće koje daje NB bile uporedive sa Linear SVC, primjenjena je sigmoidna funkcija. Na ovaj način se sirovi skorovi mapiraju u [0,1] interval i postaju uporedivi, a ujedno smo omogućili robusno kombinovanje modela. Ovaj pristup imao je više prednosti od smanjenja šuma i nerelevantnih tokena do agregiranja rezultata različitih klasifikatora.

3.2.2. Filtriranje podataka

Čišćenje i filtiranje podataka nisu ista stvar jer čišćenje uklanja šum iz teksta dok filtriranje uklanja čitave uzorke koji su neupotrebljivi. U ovaj proces podrazumijevamo eliminaciju uzoraka koji su prazni, nepotpuni, prekratki, predugi, duplicirani ili na drugi način nekonzistentni. Filtriranje se primjenjuje prije vektorizacije i balansiranja kako bi se obezbjedilo da model uči na čistim i reprezentativnim primjerima. Prazni i skoro prazni mail-ovi stvaraju rijetke vektore bez semantičke informacije i povećavaju varijansu modela i otežavaju učenje jasnih granica razdvajanja. Prekratki tekstovi ne donose dovoljno informacija za vektorizaciju i tada njihov TD-IDF bude blizu nule i model dobija konfuzan signal. Kako prekratki tako i predugi tekstovi, mogu uticati samo što kod dugih tekstova vršimo odsijecanje poruka. Na ovaj način se doprinosi stabilnijoj raspodjeli dužina i umanjuje rizik od učenja sporednih obrazaca.

```
df = df[df['body'].str.len() > 20]
df = df[df['body'].str.len() < 5000]
```

Duplikati su jako česti i mogu dovesti do optimističnih procjena performansi. Uklanjanjem duplikata čuvamo nezavisnost validacionog skupa i procjena generalizacije postaje realnija. Za svaku poruku kreirana je nova kolona body_hash koja sadrži heš vrijednost dobijenu iz normalizovanog teksta. *Heš* funkcija služi kao kompaktniji identifikator sadržaja, čime se značajno ubrzava proces poređenja velikog broja poruka. Konačno, korišćenjem metode *drop_duplicates*, uklonjene su sve poruke koje imaju identičnu heš vrijednost, a pomoćna kolona je obrisana jer više nije bila potrebna.

```
norm_for_hash = (df_all['body'].str.lower()
        .str.replace(r'\s+', ' ', regex=True)
        .str.strip())
df_all['body_hash'] = norm_for_hash.apply(hash)
df_all = df_all.drop_duplicates(subset=['body_hash']).drop(columns=['body_hash'])
```

Posljedice preskakanja filtriranja može dovesti do pogrešne metrike zbog duplikata, neadekvatnog balansiranja, nestabilne kalibracije i varijabilnih pragova odluke, a ujedno i do veće vjerovatnoće overfitting ana artefakte. Krajnji efekat filtriranja ogleda se u stabilnijem vektorskom prostoru, boljoj kalibraciji vjerovatnoća i efikasnijem radu algoritama. Manje šuma znači da TF-IDF generiše konzistentnije raspodjele težina, a to direktno pomaže linearnim modelima poput Linear SVC-a i NB-a. Istovremeno, algoritmi balansiranja dobijaju čistiji ulazni prostor za interpolaciju, pa su i generisani uzorci reprezentativniji. Osim toga, filtriranje smanjuje memorijski otisak i vrijeme obrade, što je posebno važno kod velikih korpusa. Filtriranje podataka nije samo tehnička operacija čišćenja, već predstavlja strateški korak kojim se obezbjeđuje validnost evaluacije, reproducibilnost eksperimenta i dugoročna stabilnost modela. Dosljedno provođenje ovog postupka čini razliku između sistema koji radi dobro samo na testnom skupu i sistema koji pokazuje robusne performanse u stvarnom okruženju.

3.2.3 Balansiranje podataka

Skup podataka je pokazivao disbalans između legitimnih i phishing poruka. Modeli bi težili davanju prednosti većinskoj klasi pa bi to dovelo do velikog broja lažno negativnih predikcija. SMOTE tehnika funkcioniše na taj način što sinetički generiše nove primjere manjinske klase interpolacijom postojećih tačaka u višedimenzionalnom prostoru osobina (Géron 2022). Većinska klasa na ovaj način ne trpi gubitke, a dolazi do ravnomjerne distribucije klase. Na slici 2. prikazan je disbalans između klasa jer klasa 0 koja predstavlja legitimne mail-ove sadrži 52216 uzoraka dok klasa 1 sadrži 45103 uzorka i predstavlja phishing mail-ove. Ova razlika od 7113 uzorka nam je ukazala potrebu za balansiranjem klasa što smo riješili uz pomoć SMOTE tehnike.

Slika 2. Distribucija klasa prije primjene SMOTE metode

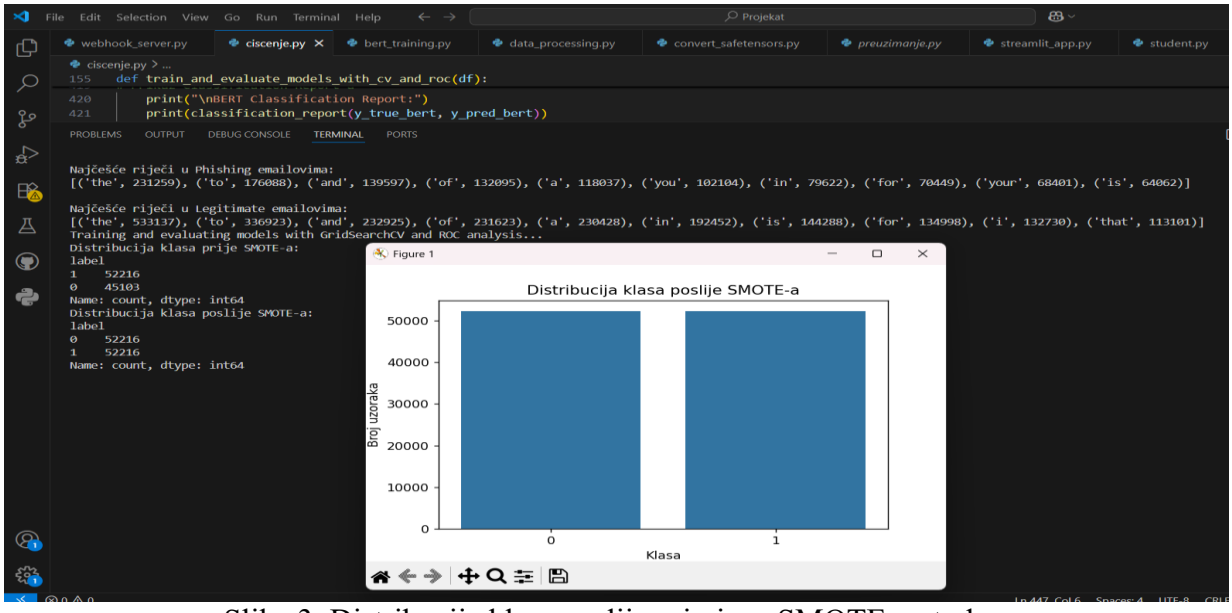
```
print("Distribucija klasa prije SMOTE-a:")

print(y.value_counts())

plt.figure(figsize=(6, 4))

sns.countplot(x=y)

plt.title("Distribucija klasa prije SMOTE-a")

plt.xlabel("Klasa")

plt.ylabel("Broj uzoraka")

plt.show()

smote = SMOTE(random_state=42)

X_resampled, y_resampled = smote.fit_resample(X_tfidf, y)

print("Distribucija klasa poslije SMOTE-a:")

print(pd.Series(y_resampled).value_counts())

plt.figure(figsize=(6, 4))

sns.countplot(x=y_resampled)
```

```
plt.title("Distribucija klasa poslije SMOTE-a")

plt.xlabel("Klasa")

plt.ylabel("Broj uzoraka")

plt.show()
```

Izvšena je vizuelna analiza distribucije klasa pomoću funkcije *value_counts()* i biblioteke *seaborn*. Ova kombinacija omogućava jasnu vizuelnu percepciju odnosa između ciljanih oznaka što je ključno balansiranje skupa podataka. U implementaciji je iskorištena funkcija *fit_resample()* iz biblioteke *imblearn* koja se primjenjuje nad TF-IDF vektorima, a rezultat ove operacije je novi skup podataka *x_resampled* i *y_resampled.*

Skup se dalje koristi u procesu treniranja i validacije modela. Nakon transformacije se vrši ponovna vizualizacija raspodjele klasa pa se time i potvrđuje primjena transformacije na konkretan skup. Ova tehnika se često koristi u zadacima detekcije prevara, prepoznavanju spam/phishing poruka i u situacijama kada su neravnoteže između klasa česte (Krawczyk 2016).

Analiza domena uključuje procjenu da li pošiljaoc koristi sumnjive domene, a phishing pošiljaoci koriste neregulisane TLD-ove(Top Level Domain) kao što su .xyz, . club, .top, .site, .online, .info, .qq, .tk. Analiza domena pošiljaoca predstavlja ključnu heurističku tehniku u otkrivanju phishing poruka, jer napadači često koriste domene sa sumnjivim nastavcima ili lažno predstavljene mail-ove koji imitiraju legitimne (Abu-Nimeh et al. 2007). Phishing domeni često kombinuju autentične ključne riječi s X nesigurnim TLD-ovima.



Slika 3. Distribucija klasa poslije primjene SMOTE metode

Na slici 3. prikazana je distribucija klasa nakon primjene SMOTE-a, gdje je broj uzoraka u manjinskoj klasi izjednačen sa većinskom klasom. Obje klase sada sadrže po 52216 uzoraka, što potvrđuje da je postignuta ravnoteža i omogućeno učenje modela bez pristrasnosti prema jednoj klasi.

3.2.4. Analiza domena pošiljaoca

Ekstrakcijom domena iz *from* polja, a potom poređenjem sa unaprijed definisanim listom visokorizičnih TLD-ova vrši se klasifikacija izvora kao sumnjivog ili legitimnog. Zlonamjerni akteri često se oslanjaju na registraciju domena sa manje kontrolisanim TLD-ovima koji su jeftiniji i slabije regulisani. Pravila su jednostavna ali imaju visoku praktičnu vrijednost pogotovo u fazama detekcije kada modeli mašinskog učenja još nemaju dovoljno konteksta. Split('a')[-a] uzima dio iza @, a ako @ nije prisutan biće vraćen prazan string.

```
def check_suspicious_domain(from_field):

suspicious_tlds = ['.xyz', '.top', '.site', '.club', '.loan']

domain = from_field.split('@')[-1] if '@' in from_field else ""

return any(domain.endswith(tld) for tld in suspicious_tlds)

df['suspicious_from'] = df['from'].apply(check_suspicious_domain)
```

Domain.endswith(tld) će biti true ako domen završava sa nekim domenom sa liste sumnjivih i *any(domain.endswith(tld) for tld in suspicious_tlds)* će vratiti true ako jedan od uslova važi. Heurističke tehnike oslanjaju se na mehanizme zasnovane na pravilima i obrasce koje definišu stručnjaci, a koji služe za identifikaciju sumnjivog ponašanja. Iako su efikasne u detekciji poznatih prijetnji, njihova statička priroda ograničava prilagodljivost u suočavanju sa sve razvijenim strategijama. Ova metoda je zasnovana na „pravilu palca" i koristi se kada nema jasnog algoritma. Ona koristi pravila kao što su: sumnjivi TLD, mnogo linkova, subject sadrži riječi kao što su urgent, verify, URL-ovi imaju skraćenu formu. Prednost ove metode je što može da funkcioniše bez velikog skupa, brza i jednostavna implementacija. Zbog svoje brzine izvršavanja kao prvi sloj filtriranja se često koriste heuristike prije primjene naprednijih modela. Njihova direktna veza sa poznatim indikatorima kompromitacije čini ih vrijednim dodatkom svakom hibridnom sistemu detekcije. Uz pomoć implementacije ovih pravila omogućiti će se rano prepoznavanje rizičnih mail-ova čak i kada trenirani modeli nemaju dovoljno informacija.

```
def check_urgent_subject(subject):
urgent_keywords = ['urgent', 'account', 'verify', 'suspended', 'immediate action', 'failed']
subject = subject.lower() if subject else ""
return any(keyword in subject for keyword in urgent_keywords)
```

U ovoj funkciji se pregleda da li *subject* sadrži riječi koje ukazuju na prevaru. Skraćeni URL-ovi su česta tehnika obmane u phishing napadima jer korisniku prikrivaju kranju destinaciju linka. Upotrebom API servisa za razvijanje URL-a moguće je otkriti stvarnu adresu i analizirati njenu reputaciju (Alsharnouby, Alaca and Chiasson 2015).

```
def contains_suspicious_links(body):
  phishing_keywords = ["security", "verification", "account", "update", "confirm", "reset", "paypal", "bank", "login", "password", "urgent"]
  if any(domain in body.lower() for domain in phishing_domains):
    return True
  if any(service in body.lower() for service in skraceni_url_servisi):
    return True
  if re.search(r'https?://(?!www\.)[a-zA-Z0-9.-]*paypal|bank|security|verification[^\s]*\.[a-z]+', body, re.IGNORECASE):
    return True
  if any(keyword in body.lower() for keyword in phishing_keywords):
    return True
  return False
```

Funkcija *contains_suspicious_links(body)* implementira heuristički mehanizam za detekciju phishing poruka kroz analizu sadržaja maila. Ona prepoznaje sumnjive obrasce u tekstu, uključujući poznate phishing domene, ključne riječi i posebno servise za skraćivanje URL-ova kao što su *bit.ly*, *tinyurl.com* ili *t.co*. Upotreba skraćenih linkova predstavlja jednu od najčešćih tehnika obmane jer prikriva stvarnu odredišnu adresu i time otežava korisniku da

vizuelno procijeni autentičnost web stranice. Na ovaj način napadači mogu maskirati maliciozne linkove koji vode ka lažnim stranicama za unos korisničkih podataka ili finansijskih informacija. Funkcija koristi i regularne izraze za otkrivanje URL obrazaca koji sadrže pojmove kao što su *paypal*, *bank* ili *security*, što dodatno povećava preciznost detekcije.

Funkcija kombinuje statičke metode, crne liste, URL pattern i servise za prikrivanje linkova kako bi detektovala phishing indikatore. Ovo je funkcionalan heuristički pristup, a ujedno jednostavan za detekciju phishing mail-ova koji je jako koristan kao karakteristika za mašinsko učenje. Ključni element svakog phishing napada je skriveni link koji vodi korisnika na lažne login stranice.

Slika 4. WordCloud prikaz najčešćih riječi u phishing i legitimnim porukama

Implementacija u okviru istraživanja izvršena je pomoću biblioteke WordCloud u Pythonu gdje su prethodno očišćeni tekstovi konvertovani u nizove, očišćeni od interpukcijskih znakova, tokenizovani i normalizovani, nakon čega su vizuelizovani kao grafički prikaz riječi (Bird Klein and Loper 2009).

WordCloud tehnike omogućavaju grafički prikaz najfrekventnijih termina u tekstualnim dataset-ovima, olakšavajući uvid u tematske obrasce i raspodjelu termina u različitim klasama teksta(Heimerl et al. 2014). Vizueliacija sadržaja je izvršena kroz Wordcloud analizu, a tekstualni sadržaj svih phishing i legit mail-ova se kombinuju u dvije cijeline. Legitimni mail-ovi najčešće sadrže riječi koje nisu usmjerene na izazivanju panike, a vizuealizacija koriniku omogućava da uoči razliku u retorici između dvije kategorije poruka. Ova tehnika služi kao dodatan alat u validaciji modela i edukativno utiče na samog korisnika.

Tekstovi su prvo prošli kroz fazu čišćenja, a zatim su WordCloud-ovi generisani za phishing i legitimnu klasu zasebno.

Kod prvo filtrira skup podataka tako da izdvaja sve mail poruke označene kao phishing (label == 1) i spaja njihov tekstualni sadržaj iz kolone body u jedan objedinjeni string. Ista operacija se ponavlja za legitimne poruke (label == 0) i tako se dobijaju dva odvojena tekstualna korpusa spremna za dalju obradu. WordCloud se pokazao korisnim jer omogućava brz i vizuelan prikaz najčešćih riječi, lako se implementira u Python okruženju uz pomoć biblioteka, poredi tekstove između različitih klasa i pomaže u edukaciji korisnika.

4. Implementacija klasičnih i NLP modela

Sprovedena je implementacija klasičnih modela mašinskog učenja s ciljem evaluacije efikasnosti tradicionalnih metoda u detekciji phishing prevara. Analiza je sprovedena na prethodno očišćenom i balansiranom skupu podataka, čime je osigurana konzistentnost u evaluaciji performansi modela. Implementacija je realizovana u Python programskom okruženju, pri čemu je korištena biblioteka *scikit-learn* kao okosnica za većinu algoritama, funkcija za pripremu podataka i optimizaciju hiperparametara. U obradi tekstualnih podataka primijenjena je TF-IDF vektorizacija, koja omogućava transformaciju nestrukturiranog teksta u numeričke reprezentacije pogodne za klasifikacione algoritme. Za odabir optimalnih parametara korištena je *GridSearchCV* metoda, koja u kombinaciji sa unakrsnom validacijom omogućava izbor najefikasnijih konfiguracija modela. Evaluacija je obavljena pomoću standardnih metrika performansi kao što su tačnost, preciznost, odziv i F1-score, a vizualna interpretacija rezultata postignuta je upotrebom biblioteka *Matplotlib* i *Seaborn*. Zbog prisutne neravnoteže u broju klasa u početnom skupu, korištena je tehnika SMOTE iz *imblearn* biblioteke, čime je postignuta uravnoteženost klasa i smanjena pristrasnost modela u korist dominantne klase. Ovakav pristup omogućio je robusniju i realističniju procjenu efikasnosti razvijenih klasifikacionih sistema.

NB je probabilistički klasifikator zasnovan na Bayesovoj teoremi, uz pretpostavku uslovne nezavisnosti osobina unutar svake klase. Ovaj model koristi frekvenciju riječi za donošenje odluka što ga čini brzim i efikasnim u brojnim zadacima klasifikacije. NB spada u generativne modele što znači da on precizno opisuje process generisanja podataka iz određene klase što doprinosi robusnost prim alim skupovima podataka.

Koristili smo Multinomial NB varijantu NB koja prirodno odgovara modelovanju učestalosti pojmova u dokumentu. Multinomial NB zadržava informacije o frekvencijama riječi, a u praksi demonstrira jednostavnost implementacije i računarsku efikasnost.

Opšta forma:

$$P(C_k \mid x) = \frac{P(C_k) * \prod P(x_i \mid C_k)]}{P(x)}$$

U praksi se koristi Multinomial varijanta:

$$P(x_i \mid C_k) = \frac{N_{ik} + \alpha}{N_k + \alpha V}$$

Gdje su:

-N_{ik}: broj pojavljivanja riječi i u klasi *k,*

-N_k: ukupan broj riječi u klasi *k,*

-V:veličina vokabulara,

- α: Laplace faktor (najčešće jedan).

NB model se odlikuje izuzetno niskim zahtjevima u pogledu vremena treniranja i memorijske potrošnje. U najčešćoj implementaciji kao što je MultinomialNB, vremenska složenost treniranja je linearna u odnosu na broj primjera i broj karakteristika ($O(n \times m)$), što ga čini posebno pogodnim za rad sa velikim tekstualnim korpusima i real-time aplikacijama.

Korišten je linear SVC iz *scikit-learn* biblioteke koji predstavlja linearne SVM klasifikacije pomoću *liblinear* optimizacionog algoritma. Iako se pojmovi SVM i SVC često koriste naizmjenično, u literaturi postoji jasna razlika između njih. SVM predstavlja opšti teorijski model zasnovan na konstrukciji optimalne razdvajajuće hiper-ravni, dok SVC označava njegovu praktičnu implementaciju u okviru biblioteke *scikit-learn*. U ovom radu korišten je Linear SVC, linearna varijanta SVM-a optimizovana liblinear algoritmom, koja je posebno efikasna za visokodimenzionalne TF-IDF reprezentacije tekstualnih podataka. Njegova klasifikaciona odluka se zasniva na predznaku linearne funkcije ali se može naknadno kalibrisati vjerovatnoća klase (Fan, Chang, Hsieh, Wang and Lin, 2008).

SVM klasifikator ima za cilj treniranja pronaći hišer ravan koji što bolje razdvaja pozitivne i negativne primjere i ostavlja što veći razmak do najbližih trening tačaka (Fan, Chang, Hsieh, Wang and Lin, 2008). Dokumenta koja su predstavljena TF-IDF vektorima obično stoje u prostoru velikih dimenzija pa je ovakav pristup efikasan na tekstualnim zadacima jer linearni separator može dobro funkcionisati SVM je diskriminativni model jer direktno optimizuje razdvajanje klasa. NB i SVC modeli se razlikuju ne samo u pristupu klasifikaciji, već i u vremenskoj i prostornoj složenosti implementacije. S druge strane, Linear SVC je zasnovan na optimizacionom problemu i koristi metode iz numeričke analize za pronalaženje optimalne razdvajajuće hiper-ravni. U najgorem slučaju, vremenska složenost treniranja Linear SVC-a može biti kvadratna ili čak kubna u odnosu na broj uzoraka ($O(n^2)$ ili $O(n^3)$), zavisno od broja karakteristika i optimizatora.

Upravo zbog ove razlike, NB se često koristi kada je cilj obrada veoma velikog broja uzoraka uz ograničene resurse. Njegova osnovna pretpostavka o uslovnoj nezavisnosti karakteristika unutar svake klase često nije realistična, naročito u složenijim zadacima. Zbog

toga, iako se NB brzo trenira, njegova prediktivna tačnost u kompleksnijim klasifikacionim problemima obično zaostaje za naprednijim modelima poput SVM-a.

Teorijska razlika između NB i SVC modela takođe proizlazi iz njihove temeljne filozofije. NB je generativni model koji uči zajedničku distribuciju P(x, y), što uključuje modelovanje uslovne vjerovatnoće P(x|y) i marginalne vjerovatnoće P(y). On pokušava da objasni kako se podaci generišu iz svake klase, a zatim donosi odluku o klasifikaciji na osnovu te informacije. Suprotno tome, Linear SVC se fokusira direktno na granicu između klasa. On uči P(y|x) bez potrebe da modelira distribuciju ulaznih podataka. Ovakav pristup omogućava veću fleksibilnost i preciznost, naročito kada su podaci visoko dimenzionalni i kada postoji značajna korelacija među ulaznim karakteristikama. Funkcija odlučivanja:

$$f(x) = w^T x + b$$

Izbor između NB i SVC zavisi od primjene jer je NB odličan za brzu i efikasnu klasifikaciju u realnom vremenu, dok SVC pruža bolja rješenja u složenijim klasifikacionim scenarijima sa boljom generalizacijom na nepoznate podatke.

4.1. Primjena TD-IDF tehnike

Term Frequency–Inverse Document Frequency predstavlja mjeru značaja riječi u jednom dokumentu u odnosu na cijeli korpus dokumenata. Nju čine dva elementa TF i IDF, odnosno terminska frekvencija i inverzno dokumentna frekevencija. TF reflektuje koliko se često određeni termin pojavljuje u dokumentu dok IDF umanjuje težinu termina srazmjerno učestalosti njegove pojave u ostalim dokumentima korpusa. Intuicija iza TF-IDF jeste da će visoku težinu dobiti oni termini koji se često pojavljuju u datom dokumentu ali su rijetki u ostaim dokumentima dok riječi koje su opštepristune u našem jeziku će dobiti malu težinu.
Formula za TF-IDF (Rajaraman and Ullman 2011):

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \text{IDF}(t)$$

gdje su:

- *TF(t, d):* broj pojavljivanja termina t u dokumentu *d*,
- *IDF(t) = log(N / DF(t)),*
- *N*: ukupan broj dokumenata,
-*DF(t):* broj dokumenata koji sadrže termin *t.*

TF-IDF kombinuje kriterijume kako bi uravnotežio mjeru značaja termina. Zbog svoje sposobnosti da numerički predstavi tekst TF-IDF je postao standrdna tehnika u oblasti obrade prirodnog jezika NLP (Jones 1972). On je popularan jer poboljšava reprezentaciju podataka za

razliku od prostih modela koji ne prave razliku između informativnih i neinformativnih riječi. Vrlo česti termini mogu uticati na statistiku dokumenta ali TD-IDF prelazi ova ograničenja jer smanjuje uticaj često pristunih riječi, a ističe specifične riječi. Na ovaj način se filitrira šum u vidu čestih riječi što je korisno kod NLP zadataka. Još jedna od prednosti TF-IDF je što rezultirajući vektori imaju tendenciju da budu rijetki što omogućava lakše skladištenje i lako identifikovanje riječi koje imaju najveću težinu.

U praktičnom dijelu rada TD-IDF je realizovan korištenjem klase TF-IDF Vectorizer biblioteke *scikt-learn.* Ova klasa automatski pretvara kolekciju tekstualnih dokumenata u matricu TF-IDF karakteristika.

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(
  stop_words='english',
  max_features=2000,
  ngram_range=(1, 2)
)
```

Korišćeni su specifični parametri kao što su:

1. *Stop_words='english'* – Ovaj parametar aktivira skup engleskih stop riječi koje će biti filtrirane iz teksta pa se na taj način smanjuje dimenzionalnost i šum u podacima, a fokus ostaje na sadržajnijim terminima.
2. *Max_features=N* - Uz pomoć ovog parametra ograničava se veličina rječnika na *N* najčešćih termina u korpusu. Mi smo odabrali vrijednost *N* kako bismo imali dvostruku korist, smanjuje se memorijska zahtjevnost i potencijalno poboljšava generalizacija modela
3. *Ngram_range=(1,2)* - Uz pomoć ove postavke, pored pojedničanih riječi uzimaju se i kombinacije od dvije uzastopne riječi prilikom pravljenja rječnika. Kombinacija od dvije uzastopne riječi mogu nositi dodatne informacije jer fraze kao što su *„click here“* i *„account suspended“* mogu biti idikativnije za phishing nego same riječi izolovano. Dodavanjem ove kombinacije model dobija širi kontektst.

Navedeni parametri korišćeni su u fazi obuke modela, u kojoj TF-IDF vektorizator uči strukturu jezika i statističku zastupljenost termina na osnovu poznatih podataka. Na taj način model stiče uvid u to koje riječi i fraze najviše doprinose razlikovanju phishing i legitimnih poruka.

Nakon što je konfigurisan *TDIDF Vectorizer* sa pomenutim parametrima pozvana je metoda *fit_transform().* Uz pomoć ove metode izvršena je tokenizacija teksta, uklanjanje stop riječi, izgradnja riječnika kao i računanje normalizovanih TF-IDF vrijednosti za svaki dokument. Kao rezultat dobijena je matrica *mxNm*, gdje je *m* broj poruka, a *N* broj karakteristika. Primjer TF-IDF vektorizacije poruke na konkretnom primjeru je sledeći, poruka glasi:

„Your account has been suspended. Please verify your identity to restore full acess.“

Nakon primjene opisanih vektorizacionih metoda, tekstualna poruka se transformiše u numerički vektor koji formalno reprezentuje njen sadržaj. U slučaju da je u našem korpusu malo mail-ova koji sadrže riječi kao što su *account, suspended ili verify*, pa tada TF-IDF dodjeljuje više vrijednosti tim terminima koji se izdvajaju od većine drugih. Atributi riječi kao što su your, has, been, please biće 0 ili veoma mali.

Prednosti TF-IDF-a u identifikaciji phishing sadržaja je što poboljšava razlikovanje legitimnih i phishing poruka tako što ističe ključne riječi, računski je efikasan, doprinosi bolje performanse pri filtriranju neželjene pošte uz kombinaciju sa klasičnim algoritmima mašinskog učenja, a i njegove karakteristike su interpretabilne.

Nakon procesa obuke, trenirani TF-IDF vektor i modeli mašinskog učenja serijalizovani su i integrisani u Flask aplikaciju. Flask je korišćen kao web framework za implementaciju API servisa koji omogućava interaktivnu klasifikaciju novih poruka u realnom vremenu. U fazi inferencije, aplikacija prima sadržaj mail poruke putem HTTP zahtjeva, koristi prethodno trenirani TF-IDF vektorizator da transformiše tekst u numeričku reprezentaciju identičnim pravilima kao tokom obuke. Na taj način se obezbjeđuje konzistentnost između obuke i predikcije, a zatim prosljeđuje vector treniranim modelima. Dobijeni rezultat se potom vraća u JSON formatu.

```
import joblib

tfidf    = joblib.load("data/tfidf_vectorizer.joblib")

model_nb  = joblib.load("data/naive_bayes_model.joblib")

model_svc = joblib.load("data/linear_svc_model.joblib")

def predict_email(sender, subject, body):

   text = f"From: {sender}\nSubject: {subject}\n\n{body}"

   X = tfidf.transform([text])  # primjena istih stop_words / max_features /   ngram_range
```

```
    p_nb = float(model_nb.predict_proba(X)[:, 1][0])
    p_svc = float(model_svc.predict_proba(X)[:, 1][0])
    p_final = 0.5 * p_nb + 0.5 * p_svc
    return "phishing" if p_final >= 0.7 else "legit"
```

Kombinacija modela NB i SVC realizovana je jednostavnim ansamblom na nivou vjerovatnoća, gdje su oba modela imala ravnomjerno dodijeljenu težinu u procesu odlučivanja. Težine modela određene su heuristički jer su pokazali slične performanse na validacionom skupu. Ovakav pristup omogućava da se objedine prednosti oba modela pa je ovakav process odlučivanja izabran i radi balansa između jednostavnosti i tačnosti, čime se izbjegava favorizovanje jednog modela i postiže robusnija klasifikacija. Prag odluke od 0.7 izabran je empirijski na osnovu analize performansi tokom validacije, kako bi se postigao optimalan odnos između preciznosti i odziva te smanjio broj lažno pozitivnih klasifikacija.

DistilBERT model nije uključen u ovu kombinaciju jer koristi potpuno drugačiju arhitekturu i sopstvene jezičke reprezentacije. Uključivanje njegovih rezultata u isti TF-IDF pipeline moglo bi narušiti konzistentnost i integritet DistilBERT modela. On već ima svoj interno naučen sistem za razumijevanje jezika pa zbog toga je korišćen odvojeno, kao dodatni model za poređenje performansi i za kasniju integraciju u hibridni sistem.

4.2. Treniranje modela

U okviru implementacije modela za detekciju phishing poruka, poseban fokus je stavljen na fazu treniranja, odnosno obučavanja modela. Ova faza je ključna jer određuje kvalitet kasnije klasifikacije kao i pouzdanost predikcija. U eksperimentu su trenirana tri različita modela: Multinomial NB, Linear SVC i DistilBERT, svaki prema specifičnom režimu učenja, karakteristikama podataka i potrebama tekstualne analize. Treniranje NB je izvedeno nad već prethodno obrađenim i izabalansiranim podacima. Prvi korak je uključivao transformaciju teksta u numerički format pomoću vektorizacije, pri čemu se uzimala u obzir važnost riječi u dokumentu.

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(stop_words='english', max_df=0.7)
X_tfidf = tfidf.fit_transform(df['body'])
```

Nakon same vektorizacije primjenjeno je balansiranje uz pomoć SMOTE tehnike kako bi se izbjegla pristrasnost modela.

```
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)

X_resampled, y_resampled = smote.fit_resample(X_tfidf, df['label'])
```

Prilikom treniranja NB modela trening je realizovan uz pomoć *GridSearchCV* metode koja omogućava sistematsku pretragu najboljih hiperparametara. On osigurava da se svaki skup hiperparametara procijeni kroz više presjeka podataka, pa se dobija realnija slika o performansama modela. Na ovaj način se izbjegava oslanjanje na jedan trening i povećava generalizacijska sposobnost modela.

```
model_nb = MultinomialNB()

param_grid_nb = {'alpha': [0.1, 0.5, 1.0]}

grid_search_nb = GridSearchCV(model_nb, param_grid_nb, cv=cv, scoring='accuracy')

grid_search_nb.fit(X_resampled, y_resampled)
```

Ključni hiperparametar Multinomial NB modela je *alpha* čija je uloga da spriječi pojavu nultih vjerovatnoća za termine koji se ne javljaju u trening podacima određene klase. Na ovaj način model postaje robusniji i tolerantniji prema rijetkim riječima koje se ipak mogu javiti u realnim uslovima. Kada je vrijednost *alphe* veoma mala model postaje oštriji jer se više oslanja na učestalosti iz trening seta, ali istovremeno postaje osjetljiv na rijetke riječi i može pretjerano naglašavati razlike između klasa. Umjerene vrijednosti obezbjeđuju ravnotežu između tačnosti i generalizacijske sposobnosti, dok prevelike vrijednosti izravnavaju distribucije i dovode do smanjenja diskriminativne moći modela. Iz navedenih razloga optimalna vrijednost je određena pomoću *GridSearchCV* metode u kombinaciji sa unakrsnom validacijom. Time smo osigurali da model postiže najbolji kompromis između preciznosti i sposobnosti generalizacije. Ovaj pristup omogućava izbjegavanje overfittinga i pronalaženje optimalne konfiguracije modela. Model je postigao AUC = 0.98 i F1-score od 0.93, što pokazuje visoku sposobnost razlikovanja između klasa. ROC kriva dodatno potvrđuje njegovu pouzdanost. Sličan princip prilikom treniranja korišten je kod Linear SVC modela uz primjenu

regulacije kroz parameter *C*, kao i podešavanja maksimalnog broja iteracija. Cilj je bio da se pronađe idealna vrijednost *C* koja omogućava balans između tačnosti i generalizacije sposobnosti modela. Parametar *C* određuje strogoću modela u odnosu na pogrešne klasifikacije tokom treniranja. Kada je vrijednost *C* mala model uvodi jaču regularizaciju i dozvoljava određeni broj pogrešnih klasifikacija ali kada je *C* previše mali može dovesti do underfittinga, jer model postaje previše jednostavan i ne hvata složenije obrasce. U slučaju kada je *C* veoma veliki model nastoji da tačno klasifikuje sve trening primjere, pa se margina sužava, a rizik od overfittinga raste jer model počinje da pamti šum iz podataka. Optimalna vrijednost *C* određena je pomoću *GridSearchCV* metode kao i *alpha* kod NB. Na ovaj način Linear SVC ostvaruje visoke performanse i izbjegava ekstremna ponašanja koja bi narušila njegovu robusnost.

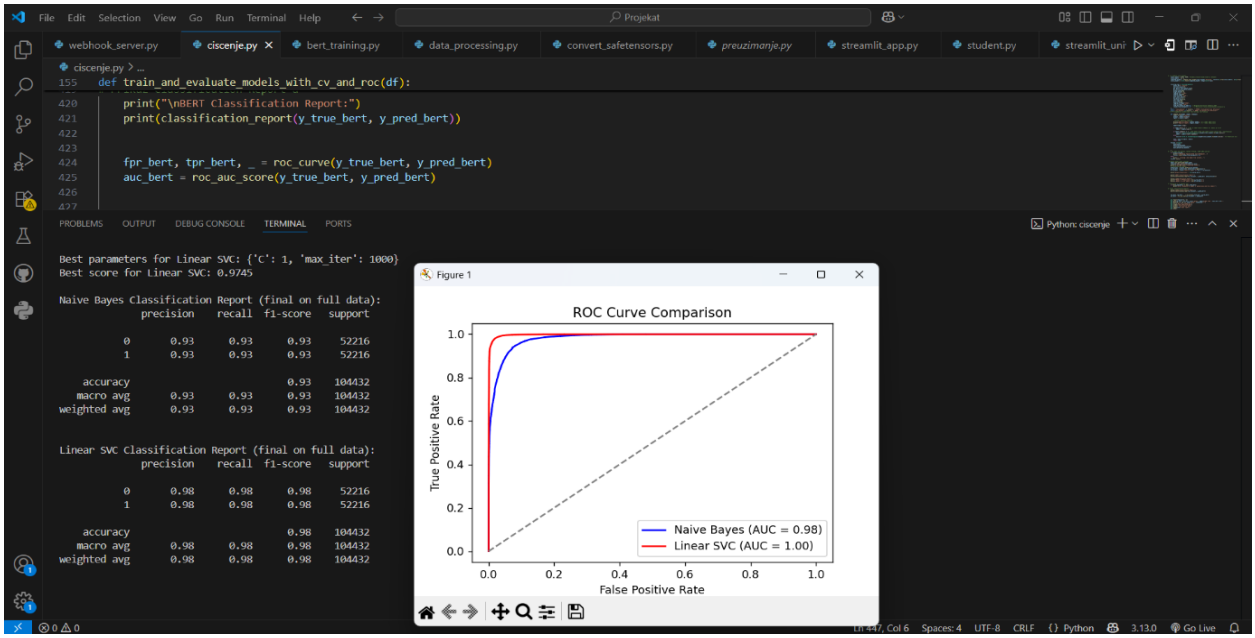
```
from sklearn.metrics import classification_report, roc_auc_score, roc_curve

y_pred = grid_search_nb.predict(X_test)

print(classification_report(y_test, y_pred))

print("AUC:", roc_auc_score(y_test, y_pred))
```

```
from sklearn.svm import LinearSVC

from sklearn.model_selection import GridSearchCV

model_svc = LinearSVC(class_weight='balanced', random_state=42)

param_grid_svc = {'C': [0.1, 1, 10], 'max_iter': [1000, 2000, 3000]}

grid_search_svc = GridSearchCV(model_svc, param_grid_svc, cv=5, scoring='accuracy')
```

Upotreba opcije class_weight='balanced' omogućila je automatsko prilagođavanje težina klasa na osnovu njihove frekvencije, što je ublažilo tendenciju modela da favorizuje češće zastupljenu klasu. Sama obuka je vršena nad prethodno vektorizovanim i balansiranim podacima, što je Linear SVC modelu omogućilo da u potpunosti iskoristi karakteristike iz TF-IDF matrice. Model je takođe evaluiran kroz ROC anallizu i *classification report* kako bi se detaljno analizirale perforamnse. Linear SVC je postigao F1-score od 0.98, uz AUC vrijednost od 1.00, što ukazuje na savršeno razdvajanje klasa. Rezultat ukazuje da je Linear SVC nadmašio NB u ovom slučaju, kako u preciznosti tako i u ukupnoj generalizaciji modela.

Vizuelno, ROC kriva gotovo dotiče gornji lijevi ugao grafikona pa samim tim pokazuje izuzetnu tačnost.

```
from sklearn.metrics import classification_report, roc_auc_score

y_pred_svc = grid_search_svc.predict(X_test)

print(classification_report(y_test, y_pred_svc))

print("AUC:", roc_auc_score(y_test, y_pred_svc))
```



Slika 5. Poređenje ROC krivih i evaluacionih rezultata za NB i SVC modele

Za oboje metode trening je izvršen pomoću *stratified k fold* metode koja je omogućila jednaku zastupljenost klasa u svakom fold-u. Classification report odnosno završni izvještaji uključuju metrike kao što su:

1. Precision – tačnost pozivitvnih predikcija
2. Recall – ispravno detektovane phishing poruke
3. F1 score – balans između precision-a i recall-a
4. Accuracy – ukupna tačnost modela

```
from sklearn.model_selection import StratifiedKFold

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

Iako je skup podataka obiman, Stratified K-Fold cross-validation je korišćen kako bi se obezbjedila stabilnija i pouzdanija procjena performansi u odnosu na jednokratnu podjelu na trening i test skup. Višestruko treniranje i testiranje na različitim podskupovima smanjuje varijansu procjene i ublažava rizik od pretreniranja na specifičnoj podjeli. Za validaciju modela korišten je cross-validation sa K=5 podskupova. Cijeli skup se dijeli na pet foldova pri čemu u svakoj iteraciji četiri folda služe za trening, a jedan fold za testiranje. Proces se ponavlja pet puta tako da svaki fold jednom bude testni, a rezultati se objedine radi realnije procjene performansi. Parametar stratified obezbjeđuje da svaki fold sadrži približno isti odnos phishing i legitimnih poruka kao i originalni skup podataka, čime se sprječava narušavanje balansa klasa.

Parametri shuffle=True i random_state=42 omogućavaju da se podaci prije podjele nasumično permutuju, pri čemu se koristi fiksno sjeme radi reproduktivnosti rezultata. Na taj način se obezbjeđuje da su foldovi reprezentativniji i da raspodjela podataka ne zavisi od njihovog izvornog redosleda. Primjena petostruke podjele predstavlja uravnoteženo rješenje između računske složenosti i tačnosti procjene performansi modela.

Konačna verzija modela trenirana je na kompletnom balansiranom skupu podataka sa optimalno pronađenim parametrima, što je rezultiralo izuzetno visokom klasifikacionom sposobnošću. Posebno je značajno to što Linear SVC model može iz težina modela direktno uočiti koji termini i obrasci najviše doprinose detekciji phishing poruka.

Na osnovu dobijenih rezultata NB je prikazao solidne performance, dok Linear SVC bolje upravlja višedimenzionalnim vektorima karakteristika pa se samim tim pokazao kao pouzdaniji klasični model u ovom eksperimentu.

DistilBERT predstavlja jedan od savremenih modela iz oblasti prirodne obrade jezika, razvijen kao pojednostavljena verzija originalnog BERT modela. Temelji se na arhitekturi transformera i treniran je metodom distilacije znanja, kojom se manji model uči da oponaša ponašanje većeg, prethodno treniranog modela (Sanh, Debut, Chaumond and Wolf 2019). Uprkos značajnom smanjenju broja parametara i veličine, DistilBERT uspijeva da zadrži većinu semantičkih sposobnosti originalnog BERT-a, uključujući razumijevanje konteksta, sintakse i značenja rečenica. Njegova arhitektura omogućava dvosmjernu analizu konteksta, što je posebno korisno u identifikaciji suptilnih obrazaca. Zbog svoje efikasnosti, brzine i preciznosti, DistilBERT se smatra pogodnim izborom za implementaciju u produkcionim okruženjima gdje su brzina i skalabilnost ključni faktori (Wolf et al. 2020). DistilBERT model je treniran u fast mode režimu, gdje je korišteno 20% podataka iz trening i validacionog skupa.

Ovaj pristup je omogućio brzu provjeru ispravnosti čitavog pipeline-a, uključujući tokenizaciju, kreiranje PyTorch dataset-a, računanje pondera klasa i konfiguraciju Trainer-a. Nakon toga je sproveden potpuni fine-tuning na 100% balansiranog skupa podataka, čime je model imao pristup svim raspoloživim uzorcima i time maksimalno iskoristio svoju sposobnost generalizacije i prepoznavanja složenih obrazaca u tekstu. Ovaj model i pripadajući tokenizator su učitani pomoću *Hugging Face transformers* biblioteke.

```
tokenizer = DistilBERTTokenizer.from_pretrained('DistilBERT-base-uncased')

model = DistilBERTForSequenceClassification.from_pretrained('DistilBERT-base-uncased', num_labels=2)
```

Ovaj dio koda inicijalizuje dva ključna elementa neophodna za primjenu DistilBERT modela u klasifikaciji teksta. Prva linija učitava tokenizator DistilBERTTokenizer, koji je odgovoran za transformaciju sirovog teksta u niz tokena koje model može da obradi. Druga linija učitava *DistilBERTForSequenceClassification* model sa izlaznim slojem za binarnu klasifikaciju, a to se postiže postavljanjem *num_labels=2.* DistilBERT zahtijeva specifičan format ulaza, uključujući *input_ids* i *attention_mask*.

```
train_encodings = tokenizer(X_resampled, truncation=True, padding=True, max_length=512, return_tensors="pt")

val_encodings = tokenizer(X_val, truncation=True, padding=True, max_length=512, return_tensors="pt")
```

Maksimalna dužina ulaza je ograničena na 512 tokena, a podaci se automatski dopunjavaju. Ovaj dio implementacije omogućava organizovanje ulaznih podataka u format pogodan za treniranje i evaluaciju *DistilBERT* modela u PyTorch okruženju.

PyTorch okruženje je okruženje zasnovano na *PyTorch* biblioteci koje se koristi za razvoj, treniranje i evaluaciju modela mašinskog učenja i dubokog učenja.

```
class EmailDataset(Dataset):
  def __init__(self, encodings, labels):
    self.encodings = encodings
    self.labels = torch.tensor(labels, dtype=torch.long)
```

Konstruktor klase prima kao ulaz već tokenizovane podatke, koje vraća Hugging Face tokenizator u formi rječnika kao i niz pripadajućih labela. Labele se konvertuju u *torch.Tensor tip,* sa long preciznošću, jer *PyTorch* očekuje labelu u takvom formatu za klasifikacione zadatke.

```
def __getitem__(self, idx):
    item = {key: val[idx] for key, val in self.encodings.items()}
    item['labels'] = self.labels[idx]
    return item
```

Metod *__getitem__* vraća jedan uzorak iz skupa podataka na osnovu indeksa idx. On uzima po jedan element iz svakog enkodiranog atributa i dodava odgovarajuću labelu, što omogućava direktnu upotrebu u Trainer petlji.

```
def __len__(self):
    return len(self.labels)
```

Metod *__len__* definiše veličinu skupa podataka, a na kraju dvije instance ovog dataset-a se instanciraju:

```
train_dataset = EmailDataset(train_encodings, y_resampled)
val_dataset = EmailDataset(val_encodings, y_val)
```

Ovdje se enkodirani trenirajući i validacioni podaci zajedno sa odgovarajućim labelama pretvaraju u PyTorch datasetove, koji se kasnije prosleđuju Trainer-u za automatizovano upravljanje treniranjem i evaluacijom modela. Ova organizacija je ključna za efikasnu obradu velikih tekstualnih skupova u NLP modelima baziranim na transformatorima. Takva implementacija prilagođenog Dataset-a osigurava potpunu kompatibilnost sa PyTorch DataLoader-om (Paszke et al. 2019). Na ovaj način se eliminiše potreba za ručnim pisanjem logike za parsiranje i dodjeljivanje labela u svakoj iteraciji trening petlje. Struktura vraćenih podataka potpuno je usklađena sa zahtjevima Hugging Face Trainer klase, pa se enkodirani tekst i pripadajuće labele automatski prosljeđuju u forward metodu modela. Ovaj pristup, ne samo da pojednostavljuje kod, već i poboljšava čitljivost i održavanje projekta, posebno kada se radi sa većim i kompleksnijim NLP skupovima podataka.

```
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=4,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=50,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    save_total_limit=2,
    learning_rate=3e-5, )
```

Ova klasa objedinjuje sve ključne opcije za treniranje i evaluaciju modela. Ukupan broj epoha kroz trening skup je jednak četiri, a što više epoha ima to se omogućava bolja generalizacija ali može dovesti i do prenaučenosti ako nije kontrolisano. U zavisnosti od memorije se koristi i broj uzoraka koji mogu poboljšati stabilnost gradijenta ako je broj uzoraka velik. Learning rate se postepeno povećava tokom početnih koraka i na taj način se omogućava stabilno treniranje i spriječavaju oscilacije u učenju modela na samom startu. Prenaučenost se ujedno spriječava uz pomoć tehnike regularizacije koja smanjuje veličinu težina u modelu, a ova tehnika je posebno korisna kod modela kao što je DistilBERT. Poređenje performansi modela omogućava se čuvanjem modela na kraju svake epohe. Zadržavaju se samo posljednja dva modela u output_dir, čime se štedi memorijski prostor. Prevelika brzina učenja može destabilizovati treniranja, dok preniska može usporiti konvergenciju, pa smo zbog toga koristili vrijednost od 3e-5 jer je empirijski potvrđena kao efikasna za fine-tunining transformer modele. Segment koda koji slijedi prikazuje pokretanje procesa treniranja pomoću Trainer klase iz biblioteke transformers.

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset
)
```

Unutar Trainer klase imamo četiri ključna argumenta, a to su:

- *model*: Instanca DistilBERT modela za klasifikaciju
- *args*: Konfiguracija hiperparametara definisana preko TrainingArguments klase
- *train_dataset:* Prilagođen PyTorch dataset sa tokenizovanim i SMOTE-bilansiranim podacima za treniranje.
- *eval_dataset*: Validacioni skup podataka

Nakon što je instanca Trainer definisana, treniranje se započinje pozivom *metode.train().* Ova funkcija automatski pokreće proces, koji uključuje forward i backward pass, ažuriranje težina, evaluaciju na validacionom skupu i logovanje metrika, u skladu s prethodno zadatim parametrima.

4.3. LoRa

Implementirali smo i low – rank adaptation(LoRa) koja predstavlja tehniku parametarski efikasnog podešavanja velikih jezičkih modela. Uz pomoć ove tehnike smo uveli dodatne trenabilne matrice niskog ranga u određene slojeve modela, a ostavili orginalne težine (Hu et al. 2022). Suština ovog pristupa je da težinski koeficijenti velikih linearnih slojeva faktorišu na proizvod dvije mnogo manje matrice. Ova tehnika se istakla u primjeni na LLM jer klasična podešavanja bi bila neizvodljiva na slabijem hardveru, a u našem radu smo koristli Loru kako bismo prilagodili DistilBERT model za zadatke uz minimalnu potrošnju resursa. Kako bismo implementirali LoRu koristili smo biblioteku *Hugging Face PEFT* (Parametar Efficent Fine Tuning) koja omogućava gotove apstrakcija za dodavanje LoRa na postojeće modele (Aathilakshmi et al. 2024). Da se ne bi trenirao kompletan model, ova biblioteka omogućava podešavanje samo manjih dodatnih komponenti kao što je LoRa. On se integriše u

postojećim transformers funkcionalnostima i podržava rad u lokalnom i distribuiranom okruženju. Korišćenjem PEFT-a u ovom projektu omogućilo nam je treniranje LoRa adaptera na osnovnom Bert modelu. U sklopu skripte LoRa.py precizirali smo ključne hiperparametre unutar klase LoRaConfig. Ključni hiperparametri su:

1. Rang adaptacije - predstavlja dimenziju skrivenog niskorangiranog prostora, u našem kodu *r* je 8;
2. Faktor skaliranja – skalar kojim se skaliraju proizvedeni LoRa izlazi i uz pomoć ovog hiperparametra se efektivno doprinosi da se LoRa slojevi usklade sa opsegom težina. LoRa_alpha je u našem radu 16, što predstavlja umjerenu vrijednost;
3. Dropout stopa – predstavlja stopu izostavljanja koja se primjenjuje u LoRa adaptaciji tokom treninga, a što je manji dropout doći će do srpiječavanja prenaučenosti adaptera;
4. Meta podešavanja ciljanih slojeva – lista slojeva unutar modela na koje će se primjeniti LoRa adapteri. Kod DistilBERT-a interna implementacija attention mehanizma se ostvaruje kroz linearne slojeve imenovane kao *query (g_lin) i value(v_lin)* pa naša LoRaConfig cilja upravo te module u svakoj od transformer blokova DistilBERT modela. Na taj način LoRa dodaje adaptere na dvije najznačajnije matriče unutar svakog attention sloja;
5. Tip zadataka – ovo osigurava da Peft pravilno integriše LoRa adaptere za scenariji klasifikacije.

```
LoRa_cfg = LoRaConfig(
  task_type=TaskType.SEQ_CLS,
  r=8,
  LoRa_alpha=16,
  LoRa_dropout=0.05,
  target_modules=['q_lin', 'v_lin'],
  bias='none'
)
model = get_peft_model(base, LoRa_cfg)
```

Nakon definisanja *LoRaConfig*, pozivamo funkcije *get_peft_model* koja interno konstruiše novi model. Svi orginalni parametri DistilBERT-a se postavljaju da ne raučunaju gradijent dok se parametri unutar LoRa slojeva označavaju kao trenabilni. Na ovaj način model ima veoma mali broj parametara koji će se mjenjati tokom obuke, pa nakon dodavanja LoRa adaptera model ima 0,93% trenabilnih parametara u odnosu na ukupan broj parametara *DistilBERT* modela. Mi nismo zamrazavali klasifikacioni sloj jer se on trenira zajedno sa LoRa adapterima dok je ostatak trasformera zamrznut, pa model uči nove task-specific parameter kroz dva izvora. Prvi izvor je LoRa adapter unutar self attention slojeva, dok su drugi izvor težine završnog klasifikatora uz oslonac na već naučene reprezentacije unutar DistilBERT backbone mreže. Koristili smo Hugging Face *Datasets* API za učitavanje i obradu dataset-a. U skripti je definisana funkcija za tokenizaciju koja primjenjuje *tokenizer.encode* plus kako bi se svi ulazi nizovi transformisali u fiksnu dužinu sekvenci.

```
def tokenize_function(examples):
    return tokenizer(examples["text"], padding="max_length", truncation=True)
```

Nakon ove funkcije izvršen je postupak mapiranja tokenizacije koji osigurava da su svi tekstualni zapisi konvertovani u numeričke ulaze koje *DistilBERT* model očekuje. Skup podataka je podijeljen na trening i na validacioni dio koji je spreman za process fine tuninga.

```
train_size = int(0.8 * len(dataset))
val_size = len(dataset) - train_size
train_dataset, val_dataset = torch.utils.data.random_split(dataset, [train_size, val_size])
```

Skup podataka je podijeljen u odnosu 80:20, gdje je 80% podataka upotreljeno za trening, a preostalih 20% za validaciju modela. Ovakva podjela omogućava da model uči iz većine dostupnih podataka, dok se na zasebnom validacionom dijelu procjenjuje njegova sposobnost generalizacije i izbjegava pretreniranje. Koristili smo maksimalnu dužinu skevence jednaku maksimalnoj dužini modela odnosno 512 tokena.

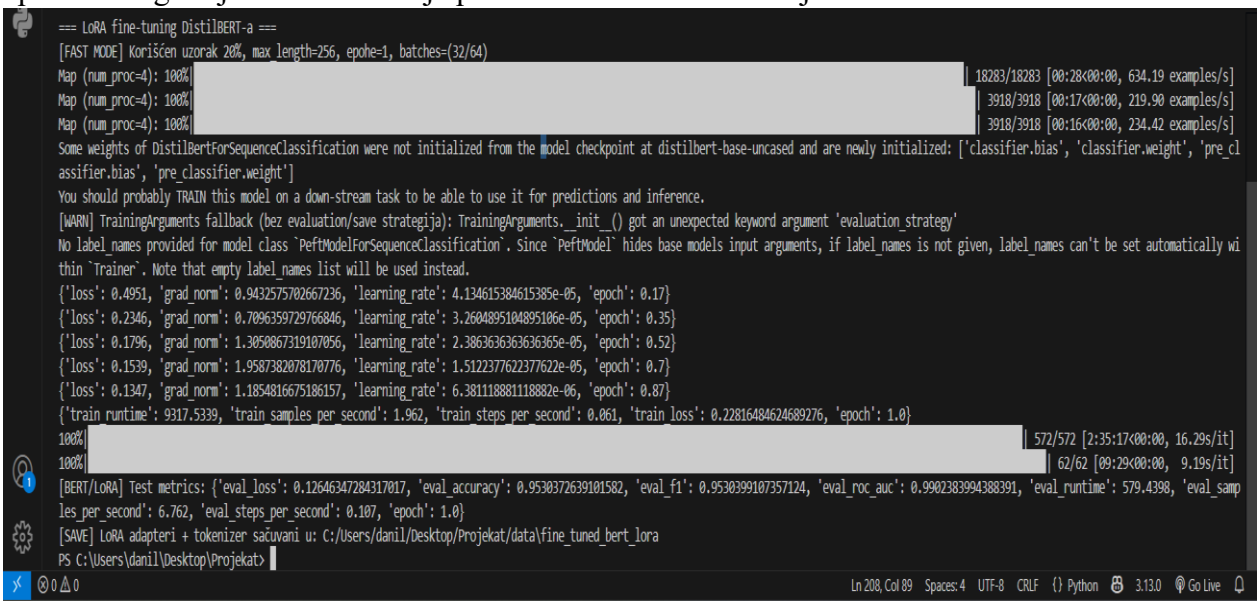
```
tokenizer = DistilBERTTokenizerFast.from_pretrained(base_model)
def tok_fn(batch):
 return tokenizer(batch['text'], truncation=True, padding=False, max_length=max_length)
ds = ds.map(tok_fn, batched=True, remove_columns=['text'], num_proc=4)
```

Jedan od ključnih izazova pri podešavanju DistilBERT modela bila je neravnoteža klasa u objedinjnom dataset-u. Budući da transformeri nisu kompatibilni sa tehnikama poput SMOTE-a, balansiranje je ostvareno primjenom class weights pristupa, gdje su težine izračunate proporcionalno učestalosti klasa i prosleđene CrossEntropyLoss funkciji. Na ovaj način smanjena je pristrasnost modela ka većinskoj klasi i obezbijeđena stabilnija evaluacija tokom treninga.

Primjena LoRa tehnike donosi više konkretnih prednosti u fine-tuningu Transformer modela. Pristupili smo prilagođavanju forward metode jer standardna implementacija *DistilBERTForSequenceClassification* u biblioteci *Transformers* ne omogućava direktno prosleđivanje tih težina kroz svoj interfejs. Unutar skripte LoRa.py nadogradili smo model tako da se prilikom računanja gubitka koristi standardna PyTorch implementacija CrossEntropyLoss sa argumentom *weight*. Gubitak se računa kao:

```
loss_fct = nn.CrossEntropyLoss(weight=class_weights_tensor)

loss = loss_fct(logits.view(-1, num_labels), labels.view(-1))
```

Prije svega, dramatično se smanjuje računarska i memorijska zahtjevnost jer se trenira svega nekoliko stotina hiljada parametara umjesto desetina miliona, koliko ima puni DistilBERT model. Obuku možemo sprovesti i na hardverski skromnijim uređajima jer je potrebno znatno manje memorije. Vrijeme treniranja je takođe kraće jer je broj operacija proračuna gradijenata i ažuriranja parametara višestruko manja.



Slika 6. Proces fine-tuning-a i rezultati DistilBERT modela pomoću LoRa metode

Log prikazuje završetak fine-tuninga DistilBERT-a s LoRa adapterima i evaluaciju na validacionom skupu. Ostvarena je tačnost ≈ 0,953, F1 ≈ 0,953 i ROC-AUC ≈ 0,998, što znači da odlično razdvaja phishing od legitimnih poruka uz minimalan broj pogrešnih odluka. U poređenju s Linear SVC-om (AUC ≈ 1,00) rezultati su praktično jednaki, ali uz znatno manju računarsku potrošnju zahvaljujući LoRA adapterima.

Još jedna prednost je što se izbjegava prekomjerno prilagođavanje kompletnog modela malom skupu podataka. LoRa omogućava da se iskoristi moć velikog transformera uz značajno manje resurse, što je i ključna prednost u našem radu gdje su nam resursi bili ograničeni.

4.4. Djelimičan fine-tuning

Osim LoRe sproveden je i klasičan postupak potpunog treniranja DistilBERT modela. Fine tuning podrazumijeva ažuriranje svih težinskih parametara unutar modela čime se ostvaruje maksimalan fleksibilnost u učenju za razliku od parametarski efikasnih tehnika kao što je LoRa (Raffel et al. 2020). Primijenjena je padding i truncation strategija kako bi se podaci mogli efikasno obrađivati u mini-batch-evima. Iako su pristupi LoRa i fine-tuningu tehnički različiti, dijele niz zajedničkih komponenti. U oba slučaja primijenjena je tokenizacija teksta korišćenjem *DistilBERTTokenizerFast*, kao i standardna podjela podataka na trening, validaciju i test skup. Zbog neravnoteže klasa, implementirana je *class-weighted loss* funkcija putem PyTorch-ove *CrossEntropyLoss* funkcije. Funkcije za evaluaciju modela koriste identične metrike kako bismo uporedili rezultate. Razlike između pristupa uglavnom se odnose na veličinu modela koji se trenira i način primjene parametarskih adaptacija. Iz orginalnog skupa izdvajaju se kolone subject, body i label, a subject i body se spajaju u jedinstveno polje text tako što se dodaje dvostruki prelom i čuva se signalna granica između naslova i tijela poruke što olakšava modelu da nauči različite obrasce.

```
df_bert = df[['subject', 'body', 'label']].copy()

df_bert['text'] = (df_bert['subject'].fillna('') + '\n\n' + df_bert['body'].fillna('')).str.strip()

df_bert = df_bert[['text', 'label']]
```

Skup se dijeli na 70% treninga i 30% privremenog skupa, a zatim se tih 30% dijela na validaciju i test. Validacija je potrebna za praćenje performansi tokom treniranja i podešavanja hiperparametara, dok test ostaje “nevidljiv” do kraja za nepristrasnu procjenu. Uz pomoć stratify se osigurava da proporcija klasa ostane ista u svim podskupovima dok random_state=42 donosi reproduktivnost. Parametar random_state=42 upotrebljen je radi

obezbjeđivanja reproduktivnosti eksperimenata, čime se garantuje da se ista podjela podataka i rezultati mogu ponoviti pri svakoj ponovnoj obuci modela. Vrijednost 42 nema matematičku posebnost, već predstavlja konvencionalno usvojeno sjeme u naučnoj zajednici, popularizovano kroz dokumentaciju biblioteka kao što je *Scikit-learn*.[2]

```
train_df, temp_df = train_test_split(df_bert, test_size=0.3, stratify=df_bert['label'], random_state=42)

val_df, test_df = train_test_split(temp_df, test_size=0.5, stratify=temp_df['label'], random_state=42)
```

U slučaju da ima više legitimnih nego phishing poruka standardni gubitak može zanemariti manjisku klasu, pa *compute_class_weight* računa težine obrnuto proporcionalno učestalosti klasa. Na osnovu datog koda učitava se pretrenirani DistilBERT i dodaje linearne slojeve pre_classifier i classifier za binarnu klasifikaciju. Svi slojevi moraju biti trenirani, odnosno parametri nisu zamrznuti i to je suština potpunog fine tuninga jer ujedno omogućava duboku domensku adaptaciju.

```
def compute_loss(self, model, inputs, return_outputs=False, **kwargs):
    labels = inputs.get("labels", None)
    if labels is None and "label" in inputs:
        labels = inputs["label"]
        inputs = {k: v for k, v in inputs.items() if k != "label"}
    outputs = model(**inputs)
    logits = outputs.logits
    if labels is None:
        loss = outputs.loss if hasattr(outputs, "loss") and outputs.loss is not None else None
        return (loss, outputs) if return_outputs else loss
    loss_fn = torch.nn.CrossEntropyLoss(weight=self.class_weights.to(logits.device))
```

[2] Nafis, N. (2021). *The story behind 'random.seed(42)' in machine learning*. Geek Culture. https://medium.com/geekculture/the-story-behind-random-seed-42-in-machine-learning-b838c4ac290a

```
    loss = loss_fn(
        logits.view(-1, model.config.num_labels)
    labels.view(-1))
return (loss, outputs) if return_outputs else loss
```

U implementaciji prilagođenog trenera (*CustomTrainer*) poseban akcenat stavljen je na metod compute_loss, koji upravlja načinom na koji se računa funkcija gubitka tokom treniranja modela. Prvi ključni element je primjena ponderisanog gubitka. Uvođenjem ponderisanog gubitka svakoj klasi se dodaje težina koja zavisi od zastupljenosti u skupu. Time se modelu pojačava glas manjinske klase, što pomaže očuvanju njenog recall-a i povećava vjerovatnoću da će spam biti ispravno prepoznat. Ovakav pristup balansira doprinos obje klase u procesu učenja i spriječava dominaciju većinske klase u optimizaciji modela. Nakon ponderisanog gubitka drugi važan element je korišćenje **kwargs paremtar. Novije verzije Hugging Face Trainer API-ja mogu proslijediti dodatne argumente koji u starijim verzijama nisu postojali, a korišćenjem **kwargs-a omogućava prihvatanje svih parametara bez obzira na verziju biblioteke. Na ovaj način kod će biti otproniji na promjene i zadržava kompatibilnost. Treći element koji doprinosi fleksibilnost implementacije je korišćenje *model.config.num_labels*. Primjena *model.config.num_labels* znači da se broj klasa automatski prilagođava konfiguraciji modela bez potrebe za izmjenom logike koda što je veliki benefit. Kada spojimo ova tri elementa dobijamo da je *compute_loss_metod* funkcionalan, felksibilan i prilagodljiv za realne uslove rada. Trainer upravlja kompletnom petljom učenja modela i u svakoj iteraciji podaci se obrađuju u manjim grupama. To omogućava efikasno korišćenje memorije i stabilnije ažuriranje parametara. Svaka manja grupa prolazi kroz forward pass pri čemu se ulazni tokeni propagiraju kroz sve slojeve modela i generišu izlazne vrijednosti modela. Na osnovu izlazne vrijednosti modela računa se vrijednost gubitka, u našem slučaju ponderisani gubitak, a nakon izračunavanja gubitka primjenili smo backpropagation algoritam na osnovu kog smo izračunali gradijente (Elharrouss, Mahmood and Bechiqito 2025). Nijedan sloj u DistilBERT-u nije bio zamrznut pa gradijenti idu kroz cijelu mrežu. Odnosno kreću se od ulaznih embedding slojeva preko transformer encoder blokova do završne klasifikacione glave.

Pristup potpunog fine tuning omogućava modelu da u cijelosti prilagodi svoje unutrašnje reprezentacije jezika zadatku detekcije spam-a.

```
os.makedirs(OUT, exist_ok=True)

trainer.save_model(OUT)

tokenizer.save_pretrained(OUT)

with open(os.path.join(OUT, "test_metrics.json"), "w") as f:

  json.dump({k: float(v) if isinstance(v, (int, float, np.floating)) else v for k, v in metrics.items()}, f, indent=2)

print(f"[SAČUVANO] Model i tokenizer u: {OUT}"
```

Nakon završetka procesa obuke, model i resursi se čuvaju putem metode *save_pretrained*. Ovaj format obuhvata ne samo parametre modela, već i njegovu konfiguraciju i sve elemente tokenizatora, pa se osigurava potpuna kompatibilnost sa metodom *from_pretrained* prilikom ponovnog učitavanja. Implementacija predviđa i čuvanje evaluacionih rezultata u datoteci test_metrics.json i ovaj fajl sadrži ključne metrike poput tačnosti, F1-score-a i ROC-AUC vrijednosti postignutih na testnom skupu. Njegova svrha je da dokumentuje performanse modela u trenutku obuke i služi kao referentna tačka za poređenje sa budućim verzijama modela. Struktura fajlova generisanih tokom save_pretrained procesa uključuje config.json –pytorch_model.bin i fajlove tokenizatora.

Ipak, radi potpune potvrde nalaza, sproveli smo dodatan eksperiment na punom skupu podataka i s većim brojem epoha, kao i testiranja s različitim početnim stanjima kako bi se procijenila varijabilnost performansi. Dalja analiza, uključujući matricu konfuzije, precision-recall krive i analizu pogrešnih klasifikacija je pružila dodatan uvid u ponašanje modela i potencijalne smjerove za njegovo dodatno unaprjeđenje. Uključivanje potpunog fine tuning-a je omogućilo detaljnu adaptaciju DistilBERT modela uz znatno veću računarsku zahtjevnost, a bio je ključan radi poređenja performansi i razumjevanju kompromisa između preciznosti i efikasnosti.

4.5. Full fine-tuning

Primijenjen je potpuni (full) fine-tuning DistilBERT modela na kompletnom skupu podataka, bez aktiviranja opcije Fast mode. Za razliku od ranije konfiguracije u kojoj se radilo na uzorku od 20% originalnog skupa radi bržeg testiranja hipoteza, ovdje se koristi cjelokupni skup podataka. U kodu smo uklonili ograničenja na veličinu uzorka, tako da se sva tri segmenta

trening, validacija i test formiraju iz punog skupa zadržavajući proporciju klasa pomoću stratifikacije:

```
train_df, temp_df = train_test_split(df_bert, test_size=0.3, stratify=df_bert['label'], random_state=42)

val_df, test_df = train_test_split(temp_df, test_size=0.5, stratify=temp_df['label'], random_state=42)
```

Kao početni referentni okvir korišćeni su klasični pristupi obradi teksta zasnovani na vektorizaciji i jednostavnijim algoritmima klasifikacije. Ulazni tekstovi, ograničeni na tijelo poruke, transformisani su u numeričke reprezentacije pomoću TF-IDF metode, pri čemu je broj karakterističnih termina ograničen na tri hiljade kako bismo balansirali informativnost. Zatim smo nad dobijenim vektorskim prostorom primijenili SMOTE algoritam za balansiranje klasa, što je omogućilo modelima da uče iz ravnotežnije distribucije primjera i bolje prepoznaju manjinsku klasu. Za procjenu i optimizaciju hiperparametara korišćena je stratifikovana petostruka unakrsna validacija, čime se osigurava da svaka podjela podataka zadržava izvorne proporcije klasa. Ispitivana su dva modela: NB sa podešavanjem parametra glatkoće α, te linearni SVM sa balansiranim težinama klasa i različitim vrijednostima parametra C. Za izbor najboljih konfiguracija oba modela korišten je *GridSearchCV* sa ponderisanom F1 mjerom kao kriterijumom. Po završetku obuke, sačuvani su trenirani modeli i pripadajući TF-IDF vektorizator, što nam je omogućilo njihovo kasnije učitavanje bez ponovnog treniranja. U poređenju s naprednijim modelima poput DistilBERT-a, ovi baseline rezultati služe kao minimalni prag performansi koji bi svaki složeniji model trebao nadmašiti. Za razliku od bržeg moda, gdje je broj epoha smanjen na jednu i gdje su batch veličine vještački povećane radi ubrzanja, ovdje se koristi standardan broj epoha čime se omogućava dublja optimizacija svih parametara mreže. Iako je maksimalna dužina sekvence zadržana na 256 tokena kako bi se kontrolisala potrošnja memorije i trajanje obuke, svi slojevi mreže ostaju trenirani zadržavajući suštinu potpunog fine-tuninga.

```
epochs = 3

lr = 5e-5

batch_train = 16

batch_eval = 32
```

```
max_length = 256

FAST_MODE = False
```

Jedan od ključnih elemenata koji kod čini robusnim za različita okruženja jeste posebni sloj za konstrukciju parametara treninga. Umjesto statičnog definisanja vrijednosti, implementirali smo mehanizam koji provjerava koje opcije određena verzija biblioteke transformers podržava i na osnovu toga automatski izostavlja nekompatibilne stavke. Na taj način izbjegavaju se ručni zahvati prilikom prelaska na novu verziju biblioteke. U proces je ugrađen i mehanizam ranog prekida treninga, čiji je kriteriji vezan upravo za istu metriku koja određuje najbolji model. Takav pristup garantuje da se zaustavljanje obuke ne zasniva na promjenama u funkciji gubitka, koje ne moraju uvijek odražavati stvarnu kvalitetu modela na validacionom skupu, već na direktnoj promjeni metrike od najveće praktične važnosti. Time se smanjuje rizik od prenaučenosti i nepotrebne potrošnje resursa. Evaluacija učinka modela oslanja se na skup metrika koje zajednički daju širu sliku performansi. Pored tačnosti, računa se i ponderisani F1 kako bi se u obzir uzela neravnoteža klasa, te ROC-AUC koji mjeri sposobnost modela da razlikuje pozitivne od negativnih primjera na nivou distribucije vjerovatnoća. Trening na 100% podataka uvodi značajne promjene u dinamici optimizacije. Budući da model sada obrađuje veći broj primjera po epohi, svaka iteracija gradijentnog ažuriranja odražava širu distribuciju podataka. U kodu se zadržavaju mehanizmi poput ponderisanog gubitka radi balansiranja neravnoteže klasa, ali se sada efekti tog balansiranja oslanjaju na realnu frekvenciju primjera u punom skupu, umjesto na njen procijenjeni podskup:

```
class_weights = compute_class_weight('balanced', classes=np.array([0, 1]), y=train_df['labels'].values)

class_weights = torch.tensor(class_weights, dtype=torch.float)
```

Još jedna bitna razlika odnosi se na vremensku i računarsku složenost. Trening na kompletnom skupu eksponencijalno povećava broj obrađenih tokena, što produžava trajanje procesa i zahtijeva značajno više GPU memorije. U ovom kodu implementirane su optimizacije poput gradient_accumulation_steps=4 i fp16=True, čime se smanjuje potrošnja memorije i ubrzava izvođenje bez vidljivog kompromisa u tačnosti:

```
base = {

  "gradient_accumulation_steps": 4,
```

```
    "fp16": torch.cuda.is_available(),
    "per_device_train_batch_size": batch_train,
    "per_device_eval_batch_size": batch_eval,
}
```

Takođe, integrisan je mehanizam ranog zaustavljanja (EarlyStoppingCallback) koji automatski prekida obuku ako validacioni rezultati stagniraju, što sprječava nepotrebno trošenje resursa i smanjuje rizik od prenaučenosti.

```
callbacks = [EarlyStoppingCallback(early_stopping_patience=2)]
```

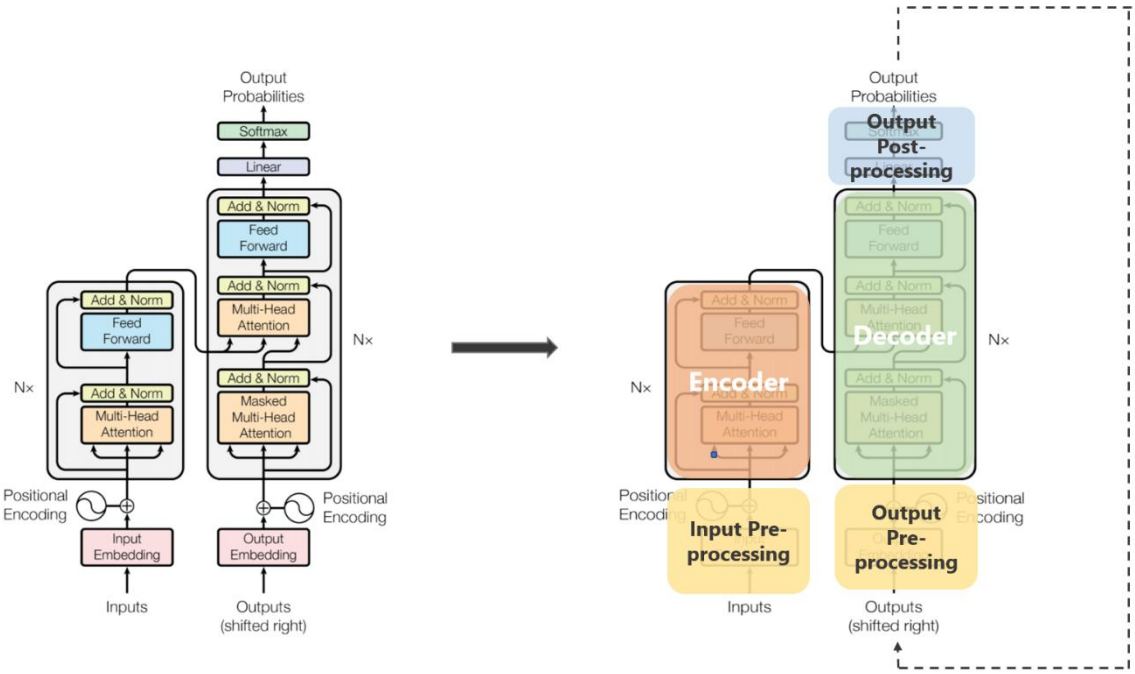
Na kraju procesa, model, tokenizer i svi relevantni metapodaci se čuvaju u standardizovanom Hugging Face save_pretrained formatu. Ovo uključuje binarne težine (pytorch_model.bin), konfiguracioni fajl (config.json), vokabular i sve pomoćne datoteke tokenizatora, zajedno sa zapisom metrika testiranja (test_metrics.json) i kompletnom istorijom obuke (log_history.json):

```
trainer.save_model(OUT)
tokenizer.save_pretrained(OUT)
with open(os.path.join(OUT, "log_history.json"), "w") as f:
    json.dump(trainer.state.log_history, f, indent=2)
with open(os.path.join(OUT, "test_metrics.json"), "w") as f:
    json.dump({k: float(v) for k, v in metrics.items()}, f, indent=2)
```

Ova tranzicija sa parcijalnog na puni skup podataka ne donosi promjene u arhitekturi modela niti u osnovnim principima obuke, ali značajno utiče na dubinu učenja, reprezentativnost naučenih parametara i pouzdanost konačnih metrika. Korištenje cijelog korpusa omogućava modelu da razvije bogatije interne reprezentacije i smanji zavisnost od slučajnih varijacija u distribuciji podataka, što je posebno važno u realnim primjenama gdje se očekuje rad sa širokim spektrom ulaznih formi.

5. Uloga velikih jezičkih modela (LLM) i primjena Mistral AI u analizi mail-a

Veliki jezički modeli (*Large Language Models*, LLM) predstavljaju pomak u oblasti obrade prirodnog jezika, omogućavajući računarima da generišu i analiziraju tekstualni sadržaj sa visokim stepenom jezičke preciznosti i kontekstualne relevantnosti. Modeli koji su trenirani na korpusima tekstualnih podataka oslanjaju se na duboke neuronske mreže kako bi predvidjeli sledeću riječ u rečenici, razumijeli značenje ulaznog teksta kao i pružili adekvatan odgovor. Značaj LLM modela naročito dolazi do izražaja u domenima gdje je potreban sofisticiran nivo semantičke obrade i kontekstualne evaluacije.



Slika 7. Standardna arhitektura Transformer modela koja predstavlja osnovu LLM modela (Rothman, 2024)

Na slici 7. je prikazana standardna arhitektura *transform* modela koja predstavlja osnovu LLM modela (Rothman 2024). Model ima dva modula odnosno encoder i decoder, pri čemu LLM koristi samo jedan od njih. Svaki sloj enkodera ima dvije komponente, a to su self attention i feed forward neuronsku mrežu. Mehanizam pažnje omogućava modelu da obrati pažnju na sve riječi u rečenici i da svakoj dodjeljuje težinu od njih, dok feed forward neuronska mreža obradi informacije i doprinsi razumijevanju značenja. Dekoder sadrži slične slojeve ali i dodatni mehanizam pozornosti prema enkoderu na osnovu kojeg se generisani tekst oslanja na kontekst iz ulaznog niza. Arhitektura koja je prikazana pruža paralelnu obradu podataka što

je ključno za skaliranje LLM. Paralelna obrada ujedno obezbjeđuje da GPU istovremeno procesuira čitave rečenice, trening bude brži, a model uči efikasnije bez potrebe da pamte prethodne tokene. U okviru ovog rada koristi se Mistral AI, savremeni LLM model otvorenog koda koji se oslanja na naprednu arhitekturu transformera. Mistral 7B predstavlja model sa sedam milijardi parametara koji je obučen za zadatke, čime omogućava korisniku da dobije objašnjenja i preporuke na osnovu kompleksnih tekstualnih ulaza. Njegova sposobnost da prepoznaje suptilne semantičke obrasce, detektuje manipulativne formulacije i formuliše evaluaciju u lokalnom jeziku i to ga čini pogodnim za zadatke automatske klasifikacije. Mistral se u ovom sistemu koristi kao dodatni sloj interpretabilnosti iznad klasičnih klasifikatora kao što su NB i Linearni SVC, te neuronske mreže poput DistilBERT modela. Nakon što je mail klasifikovan kao *junk* ili *not junk*, *Mistral* model generiše formalno objašnjenje zašto je klasifikacija izvršena na određeni način. Ovakav pristup korisniku ne pruža samo binarni ishod, već i objašnjenje koje povećava transparentnost i povjerenje u odluku sistema. Primjena Mistrala je dodatno obogaćena integracijom VirusTotal API-ja, čime model u prompt-u dobija i meta-informacije poput IP adrese pošiljaoca i njenog statusa na globalnim blacklistama. Ovo omogućava kreiranje upita koji uključuje tehničke, semantičke i sigurnosne aspekte elektronske poruke, čime se dodatno unapređuje analitička dubina odgovora koje LLM generiše (Georgi, Barham 2022). Mistral AI u ovoj aplikaciji ne funkcioniše izolovano, već kao dio šireg sistema za analizu i klasifikaciju mail-ova. Njegova uloga nije samo u generisanju teksta, već i u obogaćivanju korisničkog iskustva i pružanju dodatnog sloja bezbjednosti zasnovanog na razumijevanju prirodnog jezika i konteksta. Model je prethodno konvertovan u formu kompatibilnu sa llama.cpp i učitan lokalno pomoću sljedeće funkcije:

```
mistral = Llama(model_path=MISTRAL_MODEL_PATH, n_ctx=512)
```

Na osnovu ovog dijela koda omogućava se lokalno pokretanje modela sa ograničenim brojem tokena po upitu (n_ctx=512). Model se koristi u dvije ključne rute unutar Flask aplikacije: /test_spam i /analyze/<int:index>. U ruti /test_spam, Mistral AI se poziva nakon što se mail klasifikuje klasičnim metodama (NB, Linear SVC) i DistilBERT modelom. Prosječna vjerovatnoća da je poruka spam koristi se da bi se dobila binarna odluka (junk ili not junk), nakon čega se Mistralu šalje prompt u sljedećem formatu:

```
prompt = f""" Ti si vještačka inteligencija koja analizira sadržaj mail-ova.

Piši isključivo ijekavicom, formalno i SAŽETO.
```

```
Klasifikacija maila: **{prediction.upper()}**

Pošiljalac: {sender}

Naslov: {subject}

Sadržaj poruke:

{body}

Uputstvo:

Ako je mail klasifikovan kao *PHISHING*:

- Ukazati na elemente koji djeluju sumnjivo (ton poruke, linkovi, zahtjevi, hitnost, gramatika...).

- Objašnjenje treba jasno navesti zašto model prepoznaje rizik.

Ako je mail klasifikovan kao *LEGITIMAN*:

- Naglasiti elemente koji ukazuju na pouzdanost (neutralan ton, nedostatak linkova/zahtjeva, kontekst, uredna struktura...).

- Ne smiješ izmišljati prijetnje ili sumnjive detalje.

U svakom slučaju:

- Daj do 5 kratkih numerisanih tačaka (1.–5.).

- Ukupno do oko 150 riječi.

- Odgovor mora završiti ovako: — kraj.

"""
```

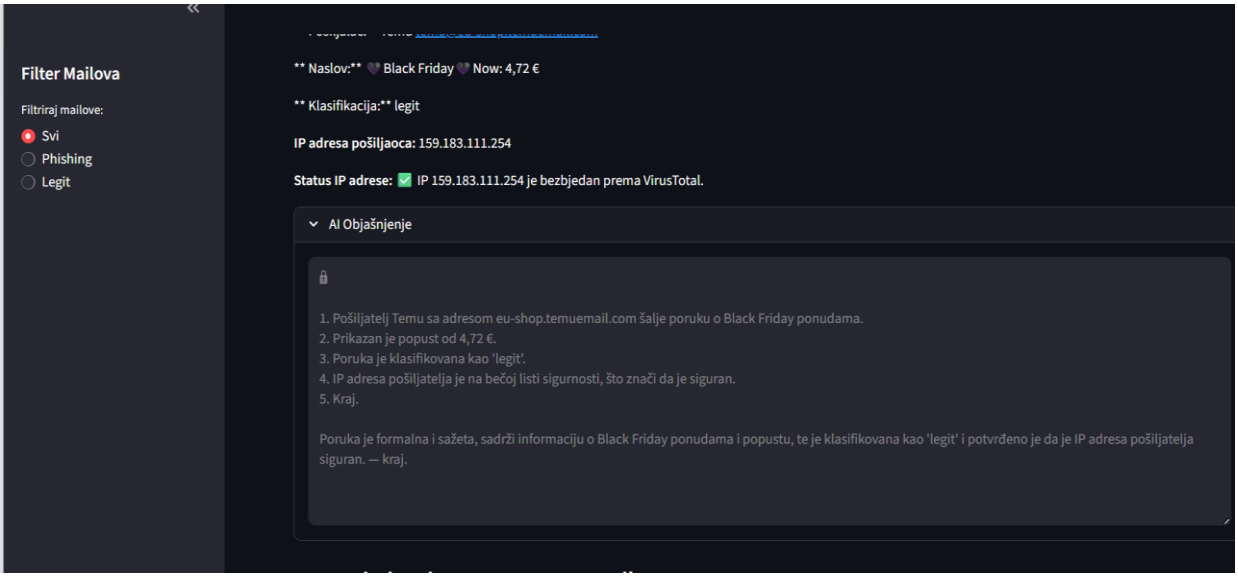
Ovaj upit kombinuje podatke kao što su pošiljalac, naslov i tijelo poruke i rezultat prethodne klasifikacije, što omogućava Mistral-u da analizira i objasni odluku. Putem funkcije *mistral* ulaz se prosljedjuje modelu, a sami rezultat se čuva i vraća klijentu u odgovoru HTTP servisa. U naprednijem slučaju rute /analyze/<int:index>, prompt uključuje IP adrese pošiljaoca prema VirusTotal API servisu. Ovdje prompt izgleda ovako:

```
prompt = (
    f"Analiziraj mail od '{sender}' sa naslovom '{subject}'. "
```

f"Klasifikovan je kao '{classification}'. "

f"Pošiljalac koristi IP adresu {sender_ip}, a status provjere na blacklisti je: {ip_blacklist_status}. "

f"Sažmi sadržaj mejla i daj preporuke kako postupiti na crnogorskom jeziku.")



Slika 8. Interfejs prototipa aplikacije za AI analizu i objašnjenje phishing mail-a

Nakon slanja upita, odgovor modela se vraća kao objašnjenje ali se ne pruža samo automatizovana klasifikacija već i transparentna i jezički relevantna intepretacija, što čini sistem pouzdanim.

Posebna vrijednost integracije Mistral modela ogleda se u njegovoj sposobnosti da prilagodi objašnjenja lokalnom jeziku korisnika, čime se povećava pristupačnost alata i istovremeno jača njegov edukativni potencijal u stvarnim uslovima primjene. Ovakav pristup potvrđuje da upotreba LLM modela prevazilazi puko generisanje teksta, te da oni mogu imati značajnu bezbjednosnu ulogu kroz interpretaciju i kontekstualizaciju identifikovanih prijetnji.

5.1. Doprinosi LLM

U ovoj implementaciji Mistral AI ne predstavlja samo generativni model za obradu teksta, već i centralnu komponentu za objašnjavanje odluka klasifikacionog sistema. Integracija sa postojećim klasifikatorima (Naïve Bayes, Linear SVC, DistilBERT) izvedena je tako da Mistral uvijek radi nakon što su svi prethodni modeli završili procjenu, čime se smanjuje rizik od pogrešnog tumačenja neobrađenih podataka. Na osnovu dobijenih rezultata iz više izvora, Mistral koristi prosječnu vjerovatnoću spam klasifikacije kao ulaznu metainformaciju, što omogućava da njegovi odgovori budu zasnovani na konsenzusu više

modela, a ne samo na jednoj procjeni. Posebno značajan aspekt ove primjene je prilagođeni prompt inženjering. Upiti su dizajnirani tako da kombinuju tehničke karakteristike, semantičke elemente te prethodnu klasifikacionu odluku. Ovakva konstrukcija ulaza povećava vjerovatnoću da će Mistral generisati objašnjenja koja su i precizna i kontekstualno relevantna. Model je optimizovan za rad u lokalnom okruženju korišćenjem llama.cpp formata (.gguf fajl), što eliminiše zavisnost od spoljnog API-ja i omogućava potpunu kontrolu nad podacima. Osim objašnjavanja Mistral implicitno obavlja i stilsku adaptaciju odgovora. Na osnovu prompta, model generiše izlaz u formalnom tonu, prilagođen lokalnom jeziku čime se povećava upotrebljivost rješenja u specifičnom jezičkom okruženju. On se ne koristi kao izolovan LLM već kao interpretativni sloj iznad višestrukih klasifikacionih modela, čime se obezbjeđuje kontekstualno svjesno objašnjavanje. Ovaj pristup ne samo da povećava povjerenje korisnika u sistem, već i olakšava otkrivanje potencijalnih lažno pozitivnih ili negativnih klasifikacija kroz dodatno obrazloženje koje može poslužiti kao osnov za reviziju odluke.

VirusTotal API predstavlja servis koji prikuplja rezultate antivirus skenera, reputacijskih baza podataka kako bi se analizirali fajlovi, domeni, IP adrese i URL-ovi u realnom vremenu. Praktični dio rada uz pomoć VirusTotal API-ja je za cilj imao procijenu rizika uz pomoć verifikacije IP adrese pošiljaoca mail-a. On nudi RESTful API koji se oslanja na HTTPS GET i POST metode, a pristup se vrši putem API ključa. API ključ se dobija nakon što se korisnik registruje na platformu i autentifikacija se obavlja tako što se u HTTP zaglavlju šalje API ključ pod imenom x-apikey.

U okviru realizacije ovog rada, VirusTotal API je iskorišćen kroz sledeći dio koda:

```
VIRUSTOTAL_API_KEY = "effc023cc60648b200ef77ed3391ebece81cac11be0d38867f2bef35687c0936"

url = f"https://www.virustotal.com/api/v3/ip_addresses/{ip_address}"

headers = {"x-apikey": VIRUSTOTAL_API_KEY}

response = requests.get(url, headers=headers)
```

U datom kodu, dinamički se formira URL na osnovu IP adrese pošiljaoca koja je prethodno izdvojena iz mail zaglavlja. Nakon što se izvrši poziv ka odgovarajućem endpoint-u API vraća odgovor u JSON formatu koji sadrži informacije o reputaciji, uključujući broj instanci. Unutar instanci se nalaze bezbjedonosni servisi koji su klasifikovali kao malicious,

suspicious, harmless i slično. Nakon dobijenih rezultata sa analize, uključuju se u finalni objekat mail-a koji korisnik vidi u Streamlit aplikaciji koja će biti objašnjena u daljem radu.

```
"ip_blacklist_status": check_ip_blacklist(sender_ip)
```

Uloga VirusTotal API-ja u ovom radu je da rezultati direktno utiču na klasifikaciju poruke i predstavljaju heuristički sloj u odluci samog sistema. JSON odgovor VirusTotal API-ja za IP adresu na datom primjeru:


```
 "data": {
  "attributes": {
   "last_analysis_stats": {
    "harmless": 76,
    "malicious": 2,
    "suspicious": 1,
    "undetected": 19,
    "timeout": 0
   },
   "network": "8.8.8.0/24",
   "country": "US",
   "as_owner": "Google LLC",
   "last_analysis_date": 1721857312 },
  "type": "ip_address",
  "id": "8.8.8.8"
 }
```


Od ukupno 98 servisa koji su analizirali IP adresu, dva su označena kao maliciozna, dok samo jedan kao sumnjičav, pa bi korisniku bilo prikazano upozorenje. U svom odgovoru za svaku adresu se vraća last_analysis_stats koja sadrži rezultate samog skeniranja sa više

različitih antivirusnih servisa, a ključne metrike su: *malicious, suspicious, harmless i undetected*. Unutar koda je logički jedostavan uslov:

```
if malicious_count > 0 or suspicious_count > 0:
    return f" IP {ip_address} se nalazi na blacklisti ..."
else:
    return f" IP {ip_address} je bezbjedan prema VirusTotal bazi podataka."
```

Ovaj kod znači da ako postoji makar jedan sigurnosni mehanizam koji je detektovao IP adresu kao malicious ili suspicios onda se tretira kao kompromitovana. Ovakva detekcija nije samostalni klasifikator ali se koristi kao dodatni bezbjedonosni signal jer postoji mogućnost da mail sadrži neutralan tekst, a da VirusTotal označi pošiljaoca kao malicioznog. U funkciji *classify_email_voting* donosi se osnovna klasifikacija na osnovu modela ali sadrži i uslov koji koristi VirusTotal signal, pa takva integracija višeslojne logike čini ovaj sistem prilagođenim za realne bezbjedonosne scenarije.

```
if contains_suspicious_links(body) or contains_suspicious_attachment(payload):
    final_prediction = "phishing"
```

6. Rezultati i diskusija

Testiranja su izvršena na istom evaluacionom skupu, kako bi se obezbijedila fer i objektivna poređenja između različitih modela. NB je ostvario tačnost od 95.64% što je ukazalo na solidne performanse uprkos njegovoj jednostavnosti. F1 score je 0.9559 i predstavlja dobar balans između preciznosti i odziva čime se potvrđuje korisnost ovog modela u okruženjima sa ograničenim resursima.

Preciznost	Odziv	F1 SCORE	TAČNOST
0.9580	0.9540	0.9559	95.64%

Linear SVC sa tačnošću od 96.52% i f1 scorom od 0.9651 prikazao je superiornost nad NB. Ovaj model koristi optimizaciju margina između klasa, što ga čini posebno pogodnim za visoko-dimenzionalne reprezentacije kao što su TF-IDF vektori. S obzirom na to da je SVC diskriminativni model, on bolje iskorišćava strukturu podataka u poređenju sa NB pristupom.

Preciznost	Odziv	F1 SCORE	TAČNOST
0.9660	0.9642	0.9651	96.52%

ROC kriva dodatno potvrđuje superiorne performanse modela Linear SVC, što je bilo evidentno i tokom procesa obuke i evaluacije modela. Posebno je značajno što Linear SVC zadržava visoku stopu tačno pozitivnih vrijednosti, čak i pri niskoj stopi lažno pozitivnih vrijednostima, što je ključno u aplikacijama gdje su lažno pozitivne klasifikacije veoma skupe. Iako postoji mali rizik od overfittinga usljed visoke AUC vrijednosti, upotreba unakrsne validacije ublažava ovu zabrinutost i ukazuje na stabilnost modela. Linear SVC se pokazao kao robusnije i pouzdanije rješenje, dok NB ostaje solidna alternativa za brzu i efikasnu primjenu u manje zahtjevnim scenarijima.

Rezultati pokazuju da je model postigao uravnotežen F1 rezultat od 0.85 za obje klase što ukazuje na to da model jednako identifikuje i pozitivne i negativne primjere. Preciznost od 0.87 za phishing poruke (klasa 1) znači da je visok procenat modelovih pozitivnih predikcija bio tačan, dok odziv od 0.82 pokazuje koliko dobro model prepoznaje stvarne phishing poruke. Slično važi i za klasu 0 (legitimni mail-ovi), sa većim odzivom i manjom preciznošću, što može skrenuti pažnju na manji broj lažno pozitivnih slučajeva.

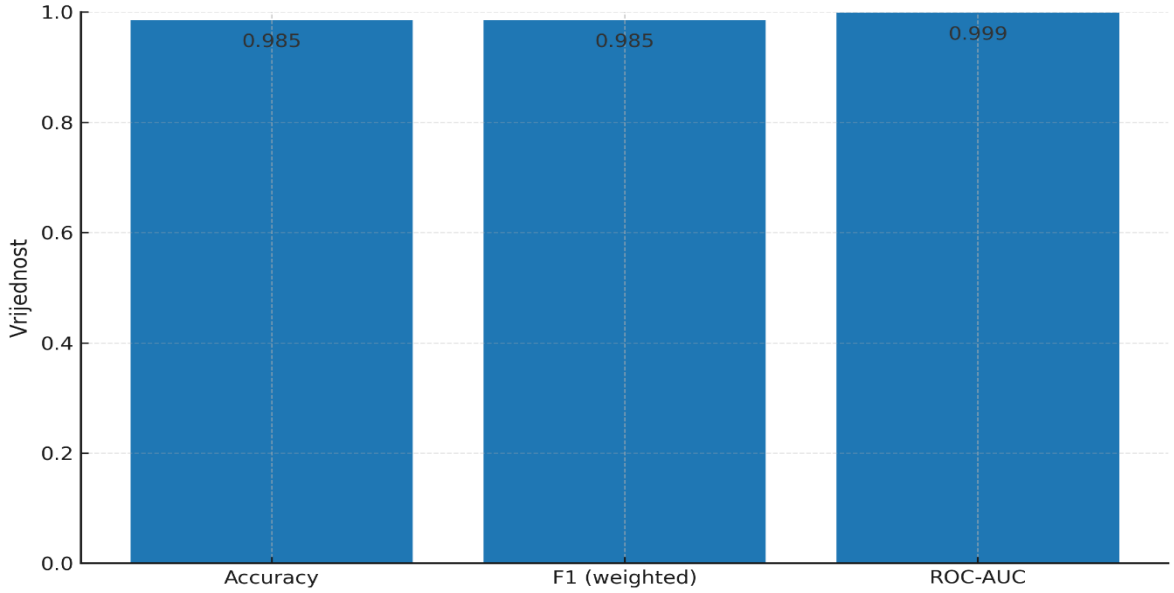
BERT Classification Report:

	Precision	Recall	F1-score	Support
0	0.83	0.88	0.85	1044
1	0.87	0.82	0.85	1044
Accuracy			0.85	2088
Macro avg	0.85	0.85	0.85	2088

Primjena LoRa tehnike na *DistilBERT* modelu rezultirala je značajnim skokom u performansama, sa tačnošću od 97.64% i F1 score-om od 0.9763. LoRa je omogućila efikasnu adaptaciju velikih jezičkih modela bez potrebe za potpunim re-treniranjem svih parametara. Rezultati potvrđuju da već i djelimično podešavanje modela može postići visok nivo tačnosti u detekciji phishing sadržaja, uz znatno manju računarsku potrošnju.

Nakon završenog treniranja postignuti su dobri rezultati na test skupu, pri čemu prosječna greška modela iznosi 0.126 što ukazuje na nisku razliku između predikcije modela i stvarnih vrijednosti, a ova vrijednost gubitka prikazuje stabilnost i tačnost modela prilikom donošenja odluka. Tačnost modela je 95.30% što znači da je model sposoban da pravilno generalizuje pravila iz trening skupa na neviđene podatke. Uravnotežen odnos između preciznosti i odziva prikazan je sa F1 scor-om i iznosi 0.953%. F1 scor prikazuje da u neravnoteženim klasama uspiješno detektuje relevantne primjere uz minimalan broj grešaka. Najbolji rezultat postignut je kroz ROC AUC metriku i iznosi 0.990 što prikazuje savršenu sposobnost modela da razlikuje pozitivne i negativne klase. Svi ovi rezultati potvrđuju da je LoRa fine tuning metodologija efikasan pristup za unaprijeđenje performansi pretreniranih jezičkih modela.

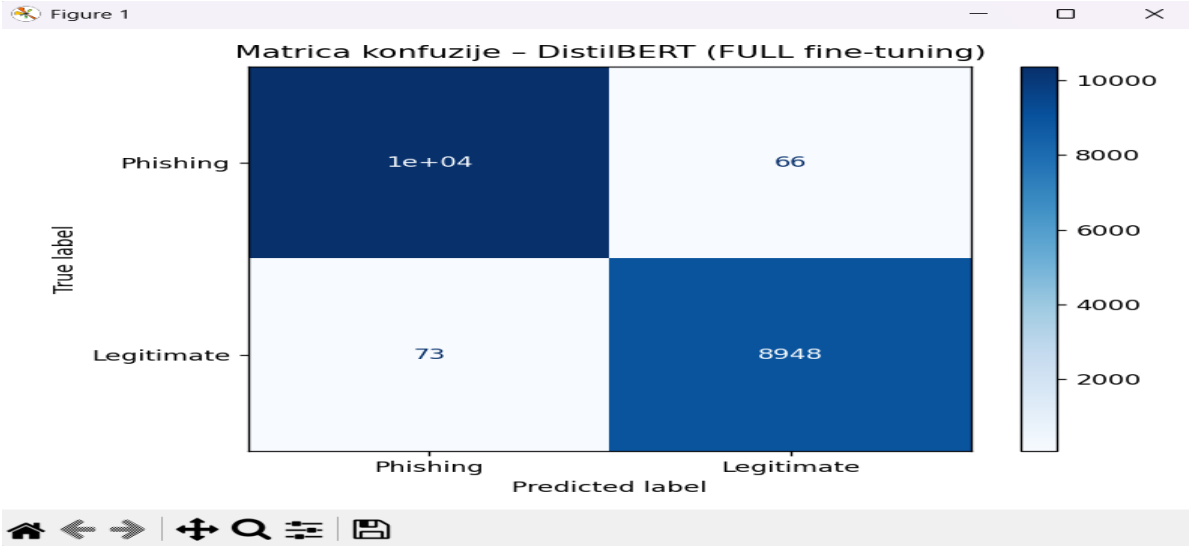
Rezultate dobijeni pri radu sa 20% ukupnog skupa podataka u režimu “Fast mode” DistilBERT modela ukazao je visoke performance. Ostvarena je tačnost od 98.5%, a ROC-AUC prikazao je rezultate od 0.999 što svjedoči o savršenoj sposobnosti modela da razluči dvije klase. Značajna osjetljivost za spam poruke se održao jer smo koristili prilagođenu funkciju gubitka koja je osigurala da doprinos primjera iz svake klase bude proporcionalno uvažen.



Slika 9. Performanse modela mjerene metrikama tačnost, F1 i ROC-AUC

Bez obzira na dobijene podatke, koristili smo i potpuni fine tuning koji je imao impresivne rezultate. Postigao je tačnost od 98.16% i sa najvišim F1 scor-om od 0.9809. S ovom verzijom se koristi kompletan kapacitet modela kako bi se naučile bogate reprezentacije mail sadržaja što dovodi do sposobnosti generalizacije na nepoznate primjere. Za razliku od drugih pristupa, ovaj zahtijeva znatno više memorijskih resursa i vremena za treniranja.

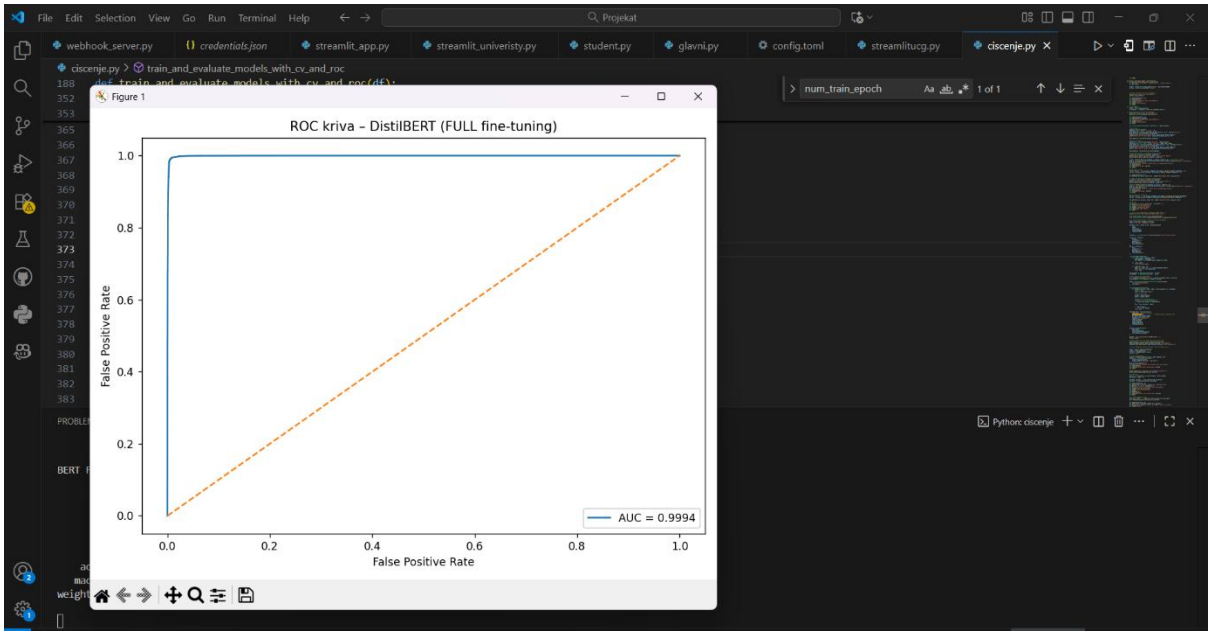
Preciznost	Odziv	F1 SCORE	TAČNOST
0.9824	0.9794	0.9809	98.16%



Slika 10. Matrica konfuzije DistilBERT-a

Matrica konfuzije prikazuje performanse DistilBERT modela nakon kompletnog finog podešavanja nad punim skupom podataka. Rezultati prikazuju sposobnost modela da razlikuje poruke, uz minimalne greške u predikciji. U gornjoj lijevoj ćeliji matrice nalazi se 10 000 tačno klasifikovanih phishing poruka, što ukazuje na visok nivo osjetljivosti modela prema malicioznim sadržajima. Broj lažnih phishing poruka koje su pogrešno označene kao legitimne iznosi 66, dok u donjoj desnoj ćeliji nalazi se 8 948 tačno prepoznatih legitimnih poruka, što potvrđuje sposobnost modela da pouzdano identifikuje sadržaj bez izazivanja velikog broja lažnih alarma. Broj legitimnih poruka klasifikovanih kao phishing, iznosi 73, što predstavlja vrlo nizak nivo pogrešne detekcije.

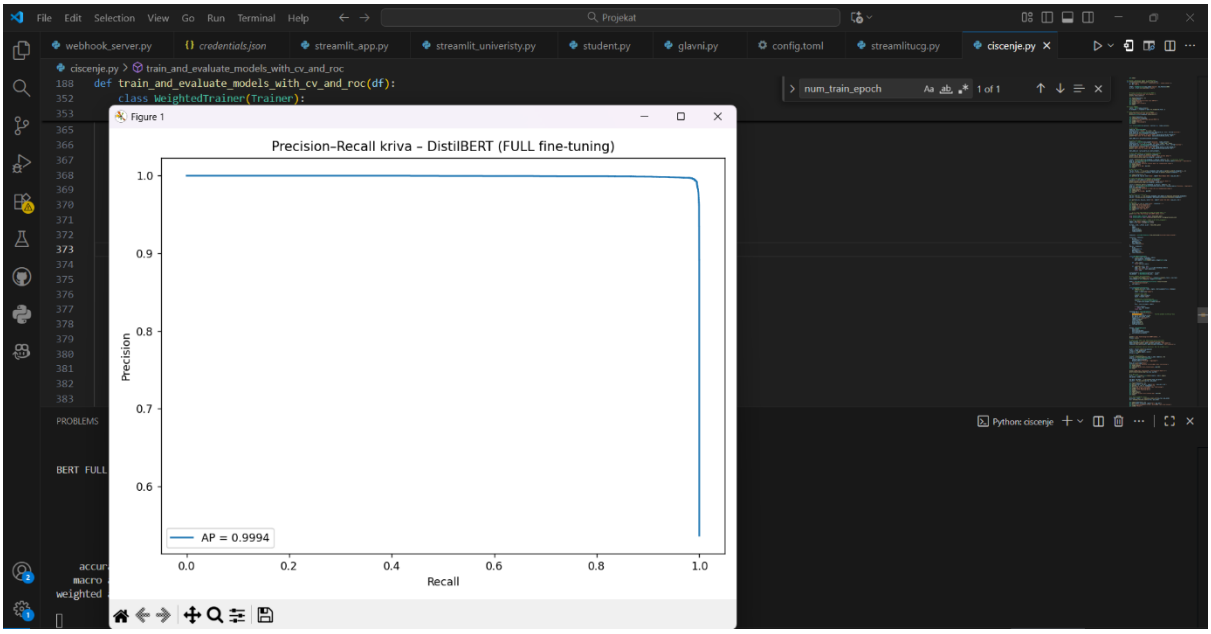
Ukupna struktura matrice pokazuje da model ostvaruje visoku ravnotežu između preciznosti i odziva, uz dominantnu koncentraciju tačnih klasifikacija duž glavne dijagonale. Distribucija grešaka ukazuje da se model pouzdano prilagodio semantičkim obrascima prisutnim u tekstualnim podacima, što je rezultat efikasne optimizacije tokom procesa treniranja.



Slika 11. ROC kriva DistilBERT modela

Površina ispod krive (AUC = **0.9994**) prikazuje da model s izuzetnom pouzdanošću dodjeljuje veće vjerovatnoće stvarnim phishing porukama nego legitimnim. Blizina ROC krive vertikalnoj osi znači da model postiže visoku stopu tačno pozitivnih detekcija i pri minimalnoj stopi lažno pozitivnih grešaka.

Dijagonalna isprekidana linija predstavlja referentni model koji predviđa slučajnim izborom, a značajno odstojanje ROC krive iznad te linije dodatno naglašava izuzetne performanse DistilBERT modela nakon kompletnog finog podešavanja.



Slika 12. Precision–Recall kriva DistilBERT modela

Kriva prikazuje kompromis između preciznosti i odziva pri variranju praga odlučivanja. Na prikazanom grafiku vidljivo je da je preciznost modela praktično konstantna i kreće se blizu vrijednosti 1.0 za širok opseg odziva, što ukazuje na to da model veoma rijetko pogrešno klasifikuje legitimne poruke kao phishing. Odziv ostaje visok do samog kraja krive, što znači da model uspješno prepoznaje najveći dio stvarnih phishing poruka. Vrijednost prosječne preciznosti je 0.9994 i potvrđuje ravnotežu između preciznosti i odziva. Tako visoka vrijednost govori o tome da model kontinuirano donosi pouzdane odluke i u situacijama kada se prag odlučivanja pomjera, što je posebno važno u praktičnim primjenama gdje se može zahtijevati strožiji ili tolerantniji režim klasifikacije.

Na osnovu logova treninga možemo da vidimo da je model prikazao brzu konvergenciju i dostigao visoke performanse već nakon prve epohe. To ukazuje na visoku informativnost ulaznog skupa podataka i dobru prilagođenost arhitekture samom problemu. Početni gubitak je iznosio 0.41 u toku prvog treninga i to je bilo očekivano za model koji se prvi put suočava sa specifičnim domenom podataka. U toku prvog dijela prve epohe gubitak je drastično pao na 0,1 i to nam je ukazalo da je model brzo prepoznao obrasce i relacije u tekstualnim podacima. Ono što smo očekivali u ranoj fazi optimizacije bila je visoka vrijednost norme gradijenata koja je bila između 2 i 6 i to nam je potvrdilo intenzivnu adaptaciju težina.

Evaluacija na kraju prve epohe pruža izuzetne rezultate, a to su: tačnost od 99,38%, F1 score od 0,9938 i ROC AUC od 0,99966. Na osnovu ROC AUC vrijednosti možemo utvridi da je model sposoban da savršeno razlikuje klase. Već u toku druge epohe gubitak se dodatno smanjio i kretao se u rasponu između 0.011 do 0.009, takođe i norma gradijenata je značajno opala što signalizira da je model ušao u fazu finog podešavanja svojih parametara, a ne velikih korekcija. Evaluacija na kraju epohe je dala tačnost od 99,44% i ROC AUC od 0.99973 što predstavlja poboljšanje u odnosu na prethodnu epohu. Na osnovu dobijenih rezultata iz druge epohe utvrdili smo da model uči brzo i da održava visok nivo generalizacije. Treća epoha nam je donijela dodatno smanjenje gubitka čak i do vrijednosti od 0.0008 dok su gradijenti praktično na nuli. To nam je bio jasan znak da je process optimizacije dosegao fazu konvergencije što znači da model više ne vrši značajne promjene u težinama. Na kraju ove epohe došlo je do blagog povećanja validacionog gubitka u iznosi od 0.0275 u poređenju sa prethodnom epohom što može ukazati na minimalnu prenaučenost. Rezultati ostaju izuzetni na kraju treće epohe, tačnost je 99.53%, F1 score je 0.9953 i ROC-AUC 0.99969. Ovako visok ROC AUC ukazuje na to da bi model bio izuzetno pouzdan čak i u situacijama kada je prag odlučivanja potrebno pomeriti, na primer u korist veće preciznosti ili većeg odziva, zavisno od specifičnih potreba sistema. Na osnovu dobijenih rezultata možemo zaključiti da je model postigao vrhunske performanse u zadatku klasfikacija teksta. Trening je pokazao napredak koji je bio brz i stabilan.

Rezultati jasno pokazuju da prelazak sa tradicionalnih klasifikatora ka dubokim modelima donosi značajno poboljšanje u preciznosti klasifikacije. Primjena LoRa tehnike pruža dobar kompromis između efikasnosti i performansi, dok potpuni fine-tuning donosi najbolje rezultate, ali po cijenu većih resursa. Ovi uvidi mogu poslužiti kao osnova za izbor modela u zavisnosti od konkretnih zahtjeva primjene. Uprkos izvrsnim rezultatima klasičnih modela, jasno je da transformerski pristupi bilo sa LoRa ili punim finim podešavanjem pružaju najbolju kombinaciju preciznosti, odziva i robusnosti. LoRa predstavlja efikasan kompromis između performansi i resursa, dok je potpuni Fine-tuning optimalan izbor kada su resursi dostupni i cilj je maksimalna tačnost. NB i SVC su i dalje veoma korisni u scenarijima gdje su brzina i jednostavnost prioriteti, ali im nedostaje sposobnost dubljeg razumijevanja jezika koje je ključno za detekciju savremenih phishing tehnika. Savremeni sistemi za detekciju phishing poruka sve češće se oslanjaju na kombinaciju više modela različitih generacija kako bi se postigao balans između brzine, tačnosti i objašnjivosti odluka. U okviru ovog rada razvijen je sistem koji integriše tri različita pristupa klasifikaciji mail poruka klasične modele mašinskog

učenja, modele dubokog učenja kao i generativni veliki jezički model koji se koristi za generisanje prirodnog jezičkog objašnjenja klasifikacione odluke. U praktičnoj implementaciji primjenjen je pristup ponderisanog glasanja pri čemu je svaki model doprinosio konačnoj odluci u skladu sa svojom pouzdanošću. Klasični modeli često gube na preciznosti u složenim tekstualnim obrascima. Zbog toga se njihov doprinos ograničava na 15% za NB i 15% za SVC, dok se 70% težine pripisuje BERT modelu, koji zahvaljujući dubokim kontekstualnim reprezentacijama pokazuje superiorne performanse u obradi jezika. Ovakva raspodjela osigurava da se zadrže prednosti jednostavnih modela u slučajevima kada su oni dovoljni, ali se odluka u većini slučajeva oslanja na sofisticiraniji pristup.

Ovakva raspodjela težina nije odabrana proizvoljno, već na osnovu empirijskih rezultata dobijenih na validacionom skupu. Modeli su evaluirani pojedinačno i na osnovu tih rezultata testirano je više kombinacija, a kao najpovoljnija se pokazala konfiguracija u kojoj su modeli ponderisani sa 0.15 za NB, 0.15 za Linear SVC i 0.70 za DistilBERT. Na taj način veća težina dodijeljena je DistilBERT-u koji pokazuje superiorne performanse, dok se doprinos klasičnih modela zadržava kao korektivni signal u situacijama kada oni bolje reaguje na jednostavnije obrasce.

```
weighted_score = (0.15 * prob_nb) + (0.15 * prob_svc) + (0.7 * prob_bert)

final_prediction = "phishing" if weighted_score >= 0.5 else "legit"
```

U pogledu performansi, testiranja pokazuju da klasični modeli imaju visoku efikasnost u obradi većih količina podataka zahvaljujući niskoj vremenskoj i memorijskoj složenosti. Međutim, njihova tačnost opada kada se susretnu sa nepoznatim obrascima phishing poruka koje odstupaju od poznatih šablona. DistilBERT kao optimizovana verzija BERT-a, omogućava analizu semantičkog značenja poruka i pokazuje značajno bolje rezultate u prepoznavanju prikrivenih prijetnji. Njegova sposobnost da razumije kontekst rečenice omogućava mu da detektuje prijetnje koje nisu eksplicitno izražene. Kombinovana strategija pokazala se efikasnijom od bilo kog pojedinačnog modela. Posebno se ističe otpornost kombinovanog sistema na neravnotežu u podacima, kao i povećana pouzdanost klasifikacije. Naime, iako pojedinačni modeli ponekad generišu lažno pozitivne ili negativne predikcije, njihova kombinacija kroz ponderisano glasanje omogućava kompenzaciju ovih slabosti, što vodi ka stabilnijim ukupnim performansama. Ovaj princip poznat je i kao ensemble learning i često se primjenjuje u savremenim AI sistemima gdje je cilj robustan učinak u realnim uslovima. Poseban doprinos sistemu daje integracija Mistral AI modela, koji ne učestvuje direktno u

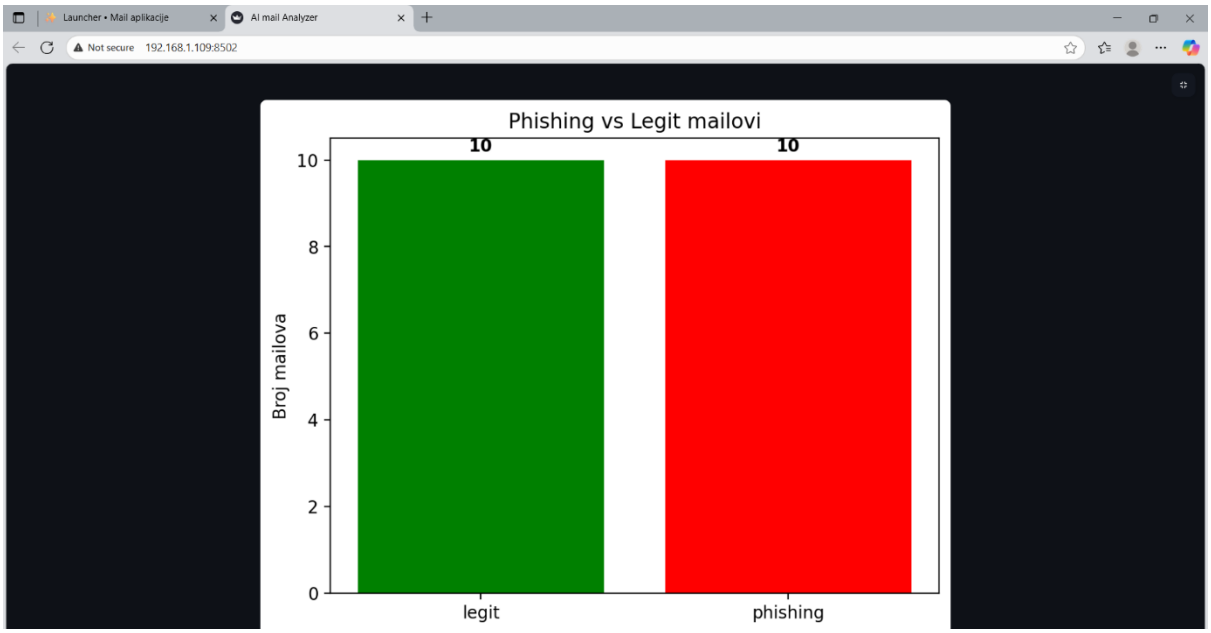
klasifikaciji, već služi kao generativni objašnjavač odluka. Nakon što klasifikatori donesu odluku, Mistral AI koristi prompt koji sadrži osnovne informacije o mail-u, njegovu klasifikaciju i meta-podatke (npr. status IP adrese na crnoj listi) kako bi generisao prirodni tekst objašnjenja na crnogorskom jeziku. Ovo objašnjenje korisniku pruža dodatni kontekst i povećava transparentnost sistema, što je naročito važno u aplikacijama koje se tiču sajber bezbjednosti. Sistem uključuje provjeru IP adresa pošiljalaca putem VirusTotal API-ja, kao i heuristike za detekciju sumnjivih linkova i ekstenzija fajlova. Ove ručno definisane sigurnosne metrike mogu djelovati kao korektivni faktor čak i ako je model klasifikovao poruku kao legitimnu, prisustvo opasnih linkova ili malicioznih ekstenzija može preinačiti odluku u phishing.

Rezultati dobijeni tokom testiranja jasno ukazuju da kombinovani sistem koji koristi klasične algoritme za inicijalnu procjenu, modele dubokog učenja za sofisticiraniju obradu teksta, i LLM za objašnjavanje odluka, ostvaruje bolje rezultate u poređenju sa bilo kojim pojedinačnim modelom. Ovakav pristup pruža balans između efikasnosti, tačnosti i objašnjivosti, što ga čini pogodnim za implementaciju u realnim bezbjednosnim sistemima.

6.1. Streamlit

Streamlit je open-source *Python* biblioteka koja omogućava programerima i data naučnicima da na brz i lak način kreiraju interaktivne web aplikacije direktno iz Python skripti. Kreirana je kako bi analitički kod i machine learning modeli bili pretvoreni u djeljive web aplikacije. Zahvaljujući spoju jednostavnosti, snage i fleksibilnosti za instalaciju biblioteke je potrebna jedna pip install streamlit komanda, a uz pomoć komande streamlit run biblioteka startuje lokalni web server i otvara aplikaciju u podrazumjevanom web pregledaču. Pozivanjem funkcija iz streamlit API-ja svi UI elementi aplikacije se definišu unutar same Python skripte. Osnov njegove arhitetkure je reaktivni model izvršavanja, odnosno aplikacija se automatski ponovo pokreće kada dođe do određenih promjena na ulazu i ne oslanja se na ručno programiranje događaje. Biblioteka pri svakoj upotrebi za osvježavanjem pokreće cijelu python skriptu ispočetka, a ova izvršavanja se dešavaju u dva slučaja. Prvi slučaj je kada dolazi do promjena na izvornom kodu aplikacije, a drugi slučaj je kada sam korisnik interaguje sa nekim widgetom u aplikaciji. Nakon pomenutih slučajeva streamlit automatski ponovo izvršava cijelukupnu skriptu pri čemu trenutna stanja postaju dostupna kao promjenjive u kodu. Streamlit koristi keširanja i čuvanje stanja da bi se omogućio efikasniji rad aplikacije bez obzira na često izvršavanje.

Streamlit ima implementiranu podršku za popularne biblioteke za vizuelizaciju kao što su *Matplotlib, Altair, Deck.gl* i druge što omogućava iscrtavanje grafika. U ovom radu primjenjen je *Matplotlib* koja omogućava korisniku da na intuitivan način sagleda distribuciju klasifikovanih poruka, čime se postiže bolja preglednost. Grafikoni doprinose bolju percepciju informacija kao i boje koje odvajaju legitimne (zelene) od phishing (crvena).



Slika 13. Distribucija phishing i legitimnih mail-ova

```
fig, ax = plt.subplots()
df["classification"].value_counts().plot(kind="bar", ax=ax, color=["green", "red"])
ax.set_title("Phishing vs Legit Email-ovi")
ax.set_ylabel("Broj email-ova")
st.pyplot(fig)
```

Aplikacija koristi *st.session state* kako bi se omogućilo očuvanje njenog stanja tokom korisničkih interakcija zbog reaktivnog ponašanja pri svakoj promjeni ulaza. Na ovaj način se spriječava ponavljanje poziva prema API-ju pri svakom osvježavanju jer se objekat st.session_state koristio za trajno smještanje preuzetih mail poruka.

```
if st.button(" Osvježi mail-ove"):
    emails = fetch_emails()
    if emails:
```

```
    st.session_state["emails"] = emails
emails = st.session_state.get("emails", [])
```

Primjenom keširanja u okviru Streamlit aplikacije omogućeno je da korisnik analizira mail-ove bez gubitka podataka iz sesije, čime se postiže fluidnije i stabilnije korisničko iskustvo. Kako bi se uštedili resursi sistema i ubrzao prikaz rezultata funkcija *fetch_emails()* je keširana na 5 minuta, odnosno ttl=300 i na ovaj način se smanjuje broj poziva prema backend servisu. Keširanjem se još eliminiše i mogućnost zagušenja servera, pa ova optimizacija postaje korisna pogotovo kod podataka koji se rjetko mjenjaju ili su vremenski stabilni.

```
@st.cache_data(ttl=300)
def fetch_emails():
  response = requests.get(API_URL)
  if response.status_code == 200:  ## HTTP status 200 označava da je zahtjev uspješno obrađen
    return response.json()
  return []
```

Komunikacija u aplikaciji se odvija putem standardizovanog REST API poziva GET I POST. Aplikacija koristi biblioteku *requests* za slanje zahtjeva ka pristupnoj tački API-ja čime se omogućava dinamičko preuzimanje podataka sa Gmail naloga, pokretanje analize pojedinačnih mail-ova kao i testiranje poruka. Na osnovu ove arhitetkure omogućava se jednostavno skaliranje i da backend logika funkcioniše nezavisno od korisničkog interfejsa. Svaka API pristupna tačka ima jasno definisanu funkciju, kao što su:

- /emails – vraća listu pristiglih mail-ova uz pomoć Gmail API-ja
- /analyze/<index> – pokreće AI analizu za odabrani mail
- /test_spam – prima korisnički definisan mail i klasifikuje ga kao spam ili legitimni

```
API_URL = "http://127.0.0.1:5000/emails"
ANALYZE_URL = "http://127.0.0.1:5000/analyze"
SPAM_TEST_URL = "http://127.0.0.1:5000/test_spam"
response = requests.get(API_URL)
```

```
requests.get(f"{ANALYZE_URL}/{email_index}")

response = requests.post(SPAM_TEST_URL, json=test_email)
```

Svaka od navedenih API pristupnih tačaka omogućava pozadinsku obradu mail-ova pomoću vještačke inteligencije. Pozivanjem */analyze/<index>* endpointa pokreće se analiza sadržaja odabranog mail-a putem više modela mašinskog učenja, uključujući NB, SVC, BERT i Mistral **(**LLM**)**.

Pomenuti modeli vrše klasifikaciju mail-a na phishing ili legitiman dok Mistral može dodatno generisati objašnjenje odluke ako korisnik želi i na taj način se implementira princip objašnjive vještačke inteligencije (XAI).

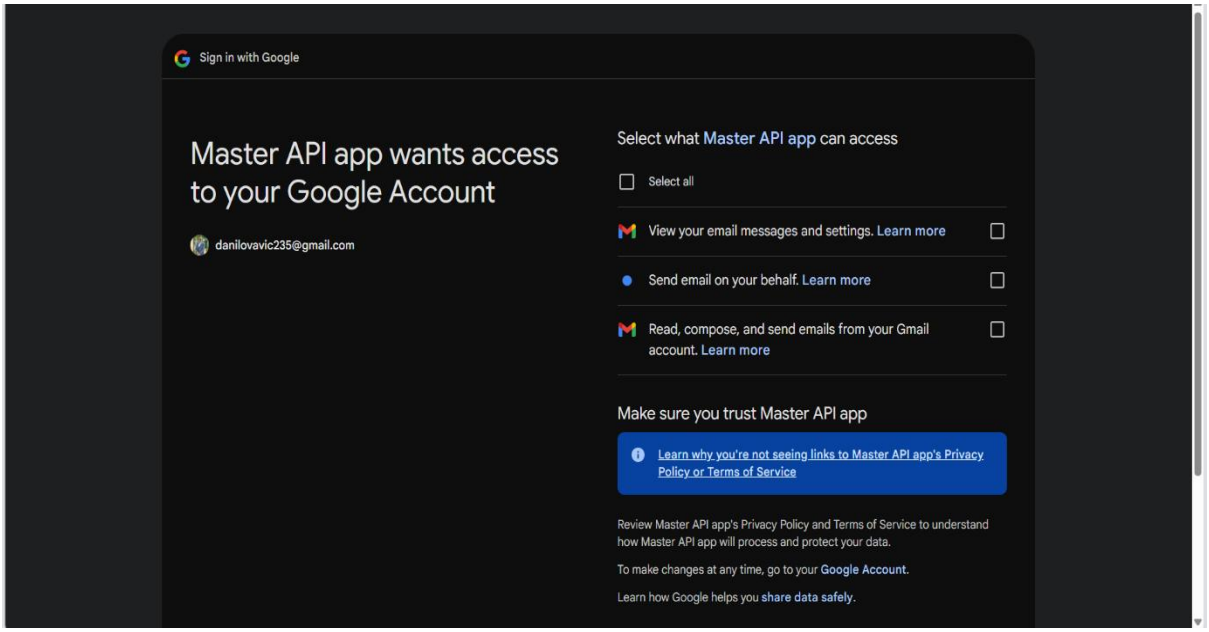
```
if st.button("Analiziraj mail"):
    analyze_response = requests.get(f"{ANALYZE_URL}/{email_index}")
    if analyze_response.status_code == 200:
        analysis = analyze_response.json()
        st.write(f"** Naslov:** {analysis['subject']}")
        st.write(f"**Klasifikacija:** {analysis['classification']}")
        if "explanation" in analysis:
            with st.expander("AI Objašnjenje"):
                st.write(analysis["explanation"])
```

Gmail API predstavlja RESTful servis koji omogućava pristup gmail nalogu korisnika na granularnom nivou, uključujući čitanje, slanje, brisanje i pretragu mail-ova, kao i manipulaciju etiketama i folderima. U praktičnom djelu rada upotrebljen je Gmail API koji je korišćen za automatizovano preuzimanje pristiglih mail-ova u realnom vremenu, što je omogućilo njihovu dalju analizu uz pomoć AI modela. Za korišćenje Gmail API potrebno je proći proces autentifikacije i autorizacije uz pomoć OAuth 2.0 protokola. Uz pomoć credentials.json fajl se ovaj process realizuje, a on sadrži ID klijenta i druge metapodatke. Da bismo pristupili API-ju bilo je potrebno napraviti projekat na Google Cloud Console i aktivirati Gmail API projekat. Nakon toga je kreniran OAuth 2.0 client ID kako bismo preuzeli

credentials.json, a inicijalizacija autorizacije je izvršena pomoću biblioteke google-auth i google-api-python-client.

```
from google.oauth2.credentials import Credentials

from google_auth_oauthlib.flow import InstalledAppFlow

from googleapiclient.discovery import build

SCOPES = ['https://www.googleapis.com/auth/gmail.readonly']

flow = InstalledAppFlow.from_client_secrets_file('credentials.json', SCOPES)

creds = flow.run_local_server(port=0)

service = build('gmail', 'v1', credentials=creds
```

Gmail API omogućava pristup resursima kao što su *messages* (pojedinačni mail-ovi), *threads*(konverzacije), *labels* (organizacione oznake) i promjenama u inboxu za sinhronizaciju. Kako bismo preuzeli mail-ove koristili smo *users.messages.list i user.messages.get* pa se prvo dobijaju ID-evi poruka, a zatim se za svaki mail povlače detalji kao što su pošiljalac, naslov, tijelo poruke i zaglavlja.



Slika 14. Prikaz ekrana autentifikacije putem Google naloga

Na slici je prikazan proces autentifikacije putem Google naloga, pri čemu aplikacija zahtijeva eksplicitnu saglasnost korisnika za pristup Gmail podacima. Ovaj mehanizam

osigurava kontrolisani i transparentan način integracije aplikacije sa eksternim servisom, čime se obezbjeđuje zaštita privatnosti i usklađenost sa sigurnosnim standardima.

```
results = service.users().messages().list(userId='me', labelIds=['INBOX'], maxResults=10).execute()

messages = results.get('messages', [])

for msg in messages:

    message = service.users().messages().get(userId='me', id=msg['id'], format='full').execute()
```

Aplikacija koristi eksplicitno inicirano ažuriranje putem REST poziva kao mehanizam kontinuirane sinhronizacije podataka. Ovaj pristup je bio dovoljan jer nam je omogućavao ručno ili automatsko osvježavanje mail-ova, izbjegavao je i potrebu za kompleksnom implementacijom google pub/sub mehanizma, a ujedno je i odgovarao logici rada streamlit aplikacije odnosno reaktivnom osvježavanju sesije. Ključna komponenta za obradu mail-ova u aplikaciji bila je korišćenje Gmail API-ja jer je omogućio dinamičko preuzimanje mail-ova, detekciju spam/phishing sadržaje direktno iz inboxa, unifikovan unos podataka i filtriranje prema rezultatima klasifikacije.

Gmail API je predstavljao ključnu komponentu u izgradnji automatizovanog sistema za analizu mail-ova, omogućio je direktno i dinamičko preuzimanje mail-ova pa se tako eliminisala potreba za ručnim unosom. Na taj način su modeli dobijali sviježe podatke direktno iz korisnikovog inboxa što je poboljšalo realističnost analize, a ujedno omogućava i pristup metapodacima na osnovu kojih su izvršene dodatne bezbjedonosne provijere kao što su detekcija sumnjivih pošiljaoca i provjera IP reputacije. Za potrebe pristupa i obrade privatnog mail-a korišćen je Gmail API, dok s druge strane za univerzitetski mail sistem implementiran je Selenium WebDriver skripta za automatizovano pristupanje studentskom nalogu, ekstrakciju poruka i skladištenje njihovog sadržaja. Upotrebom Selenium WebDriver-a moguće je automatski izvršavati korisničke akcije poput unosa lozinke i klika na dugmad, čime se simulira realna interakcija sa web aplikacijom bez potrebe za manuelnom intervencijom (Gundecha, Avasarala 2018). U ovom radu Selenium je korišten kao alat za pristup univerzitetskom mail sistemu. Poseban izazov predstavljala je obrada iframe elemenata u HTML strukturi, koja je korišćena za prikaz sadržaja mail-a, ovaj problem smo riješili uz pomoć dinamičke identifikacije iframe-ova. Korištena je funkcija *webdriver.Chrome()* koja se koristi za pokretanje Chrome browser-a, a sve operacije izvršavaju u stvarnom browser okruženju. Alat

ChromeDriverManager() eliminiše potrebu za manuelnom konfiguracijom tako što preuzima potrebne drajvere i automatski prepoznaje verziju lokalno instaliranog browser-a.

```
options = webdriver.ChromeOptions()

options.add_argument("--disable-gpu")

options.add_argument("--no-sandbox")

driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()), options=options)
```

Parametri kao što su nosandbox i disable GPU korišteni su kako bi se osigurala stabilnost prilikom pokretanja testova sa ograničenim resursima.

```
driver.get(URL)

email_input = WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "rcmloginuser")))

email_input.send_keys(EMAIL)

password_input = WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "rcmloginpwd")))

password_input.send_keys(PASSWORD)

login_button = WebDriverWait(driver, 10).until(EC.element_to_be_clickable((By.ID, "rcmloginsubmit")))

login_button.click()
```

Klasa WebDriverWait u kombinaciji sa expected_conditions se koristila kako bi se omogućilo pozudano upravljanje dinamičkim sadržajem stranice dok se HTML elementi ne pojave u DOM strukturi. Da bi se izbjegla blokada prilikom dužih analiza, implementiran je asinhroni mehanizam pokretanja analize. To je izvršeno putem Python-ove *threading* biblioteke. Na taj način se za svaki zahtjev pokreće poseban thread koji izvršava analizu i rezultat potom čuva u *job_results*. Nakon ovog klijent putem job_id identifikatora može provjeravati status izvršavanja i ovaj pristup je posebno važan za rad sa LLM-om jer analiza može trajati i više od jednog minuta.

```
emails = driver.find_elements(By.XPATH, "//table[@id='messagelist']//tr[contains(@class, 'message')]")
```

U ovom dijelu koda koristi se XPath selektor za identifikaciju elemenata unutar HTML tabele koja sadrži listu pristiglih mail poruka. Stabilnost sektora utiče na pouzdanost automatizovanih operacija jer promjene u strukturi web stranice mogu dovesti do neuspijeha u pronalaženju elemenata.

```
email_input.send_keys(EMAIL)
password_input = WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.ID, "rcmloginpwd"))
)
password_input.send_keys(PASSWORD)
login_button = WebDriverWait(driver, 10).until(
    EC.element_to_be_clickable((By.ID, "rcmloginsubmit"))
)
login_button.click()
WebDriverWait(driver, 10).until(
    EC.presence_of_element_located((By.ID, "messagelist"))
)
```

Find_element() funkcija koristi se za selekciju komponenti stranice, dok *click()* i *send_keys()* omogućavaju njihovu manipulaciju, kao što su unos korisničkog imena, lozinke i potvrda logovanja. Na ovaj način ostvarena je potpuna simulacija ljudske interakcije sa webmail sistemom.

```
email_body_text = ""
iframes = driver.find_elements(By.TAG_NAME, "iframe")
for i, frame in enumerate(iframes):
    driver.switch_to.default_content()
```

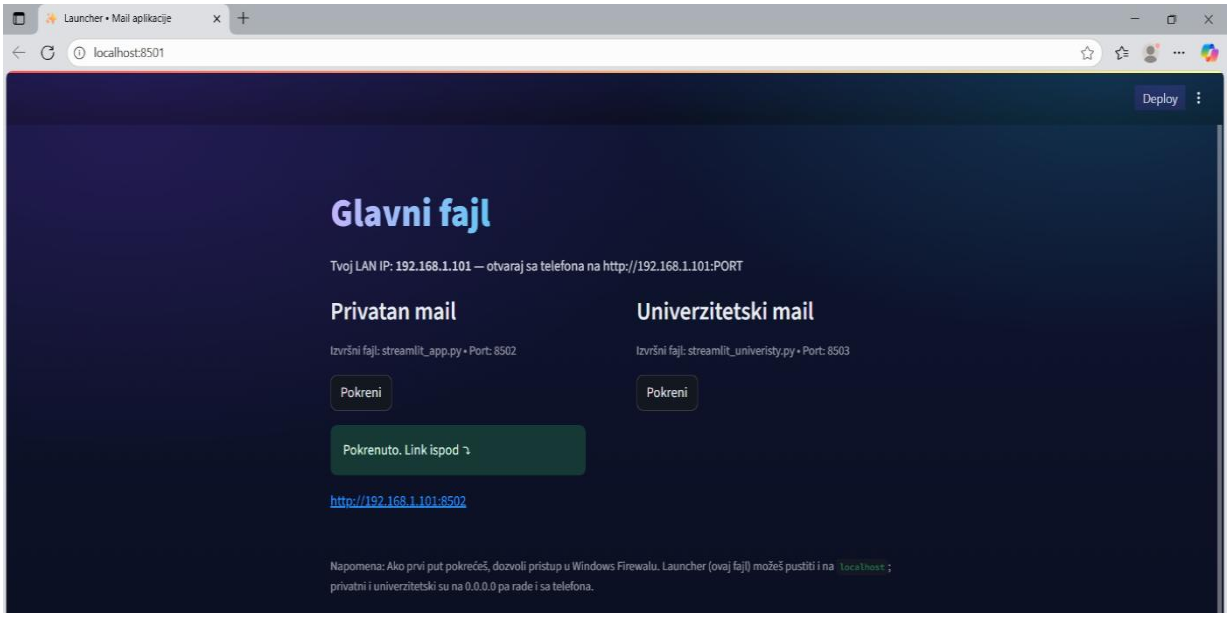
```
    driver.switch_to.frame(frame)
    try:
        email_body = driver.find_element(By.XPATH, "//body")
        email_body_text = email_body.text
```

Ovaj segment implementira postupak pretraživanja i parsiranja sadržaja mail poruke iz iframe elemenata, a korišćenjem metode *driver.switchto.frame(),* omogućeno je direktno upravljanje kontekstom iframe komponenti. Prije svakog prebacivanja, koristi se *driver.switch_to.default_content()* kako bi se resetovao kontekst na osnovni DOM. Ovaj pristup omogućava stabilno parsiranje poruka bez obzira na kompleksne strukture HTML mail-ova jer ukoliko sadržaj nije odmah dostupan, koristi se try-except blok za otkrivanje grešaka bez prekida izvršavanja.

```
email_list.append({"Naslov": email_subject, "Sadržaj": email_body_text})
print(f" mail {index + 1}/{len(emails)}")
print(f"Naslov: {email_subject}")
print(f" Sadržaj: {email_body_text[:500]}...")
print("-" * 80)
driver.get("https://mail.edu.ucg.ac.me/?_task=mail&_mbox=INBOX")
time.sleep(5)
```

Nakon ifram-a omogućeno je prikupljanje i skladištenje podataka svake mail poruke i snimaju se kao python riječnik koji sadrži njen naslov i sadržaj. Na ovaj način se omogućava jednostavna struktura za kasniju obradu, a zbog bezbjedonosnih razloga ograničili smo prikaz sadržaja na 500 karaktera. U ovom radu upoređene su dvije implementacije sistema za analizu mail poruka, verzija koja koristi Selenium za pristup univerzitetskom webmail sistemu i verzija koja koristi zvanični Gmail API. Iako predstavljaju različite implementacione pristupe, obje verzije usmjerene su ka istom cilju, a to je detekcija phishing poruka i objašnjavanje njihove prirode putem AI modela. Ove dvije verzije funkcionišu na sličan način ali ne koriste iste alate jer za univerzitetski mail je korišten Selenium koji je omogućio automatizaciju korisničkih

akcija u web okruženju, dok Gmail API omogućava direktan pristup podacima preko JSON odgovora. Koriste istu klasifikaciju na osnovu više modela dok AI analiza se vrši putem Mistral modela, dok informacije o IP adresi i statusu na blacklistama dobijaju uz pomoć VirusTotalAPI-ja.



Slika 15. Launcher aplikacije za izbor privatnog i univerzitetskog mail servisa

Ovaj Streamlit na slici 15. funkcioniše kao launcher aplikacija koja korisniku nudi centralni interfejs za pokretanje različitih modula, privatnog i univerzitetskog mail-a. Nakon izbora željenog modula, aplikacija ga pokreće na odgovarajućem portu i generiše link preko kojeg mu se može pristupiti. Od posebnog značaja bila je i mogućnost pristupa sa različitih uređaja unutar iste mreže jer se na taj način omogućilo testiranje interfejsa i funkcionalnosti na mobilnim uređajima, tabletima ili računarima. U okviru sistema omogućili smo da streamlit aplikaciji se pristupi putem IP adrese računara na kom je aplikacija pokrenuta. Da bismo to omogućili ključni tehnički preduslov za ovakvu funkcionalnost bilo je bindovanje aplikacije na adresu 0.0.0.0, dok podrazumjevana konfiguracija streamlit servera koristi 127.0.0.1. Time se pristup ograničava isključivo na lokalni računar, a podešavanje na 0.0.0.0 omogućava da server sluša zahtjeve na svim mrežnim interfejsima uređaja. Aplikacija na ovaj način postaje dostupna ne samo na lokalnom hostu već i putem mrežne kartice povezane na mrežu. Implementirali smo funkciju za dohvat lokalne IP adrese računara unutar mreže i ta adresa je dodjeljena od strane rutera, a predstavlja jedinstveni identifikator uređaja u LAN okruženju. Pored ovih tehničkih aspekata, važno je naglasiti i sigurnosne implikacije. Otvaranjem servera na 0.0.0.0 pristup postaje moguć svim uređajima u mreži, što je poželjno u okviru kućnog okruženja ali može predstavljati sigurnosni rizik u otvorenim ili javnim mrežama. Prednost

ovakvog pristupa ogleda se u jednostavnosti testiranja korisničkog iskustva na različitim uređajima, bez dodatnih troškova za servere ili kompleksne mrežne konfiguracije. Na taj način računar efektivno funkcioniše kao lokalni server, dok mobilni uređaji služe za verifikaciju responzivnosti i funkcionalnosti aplikacije u realnim uslovima korišćenja (Goodfellow 2016).

Jedna od ključnih funkcionalnosti aplikacije jeste mogućnost da korisnik postavi pitanje AI asistentu o konkretnom mail-u. Ova komponenta je uvedena sa ciljem da se poveća interaktivnost sistema i da korisnik ne ostane samo na nivou „phishing/legit“ klasifikacije, već da dobije i kontekstualno objašnjenje odluke (Brown et al. 2020). Kada korisnik postavi pitanje, aplikacija sakuplja kontekst mail-a naslov, tijelo, klasifikacija modela i samo pitanje korisnika. Ovi podaci se grupišu u JSON strukturu i šalju ka backend API-ju:

```
payload = {
   "question": user_question,
   "subject": analysis["subject"],
   "body": analysis["body"],
   "classification": analysis["classification"]
}
ai_resp = requests.post(ASK_AI_URL, json=payload)
```

Ovdje se jasno vidi da se pored pitanja korisnika prenosi i kontekstualna informacija o mail-u. To znači da AI model ne odgovara nasumično već ima na raspolaganju puni sadržaj mail-a, njegov naslov, kao i već prethodno izračunatu klasifikaciju. AI model se koristi u ulozi asistenta za objašnjenje: on ponovo ne klasifikuje mail već interpretira zašto je klasifikacija takva, koji termini ili obrasci ukazuju na phishing, kao i šta korisnik treba da uradi. Na primer:

- Ako mail sadrži fraze tipične za prevare *(„verify your account“, „click here“*), AI asistent može to eksplicitno istaći.
- Ako je IP pošiljaoca na crnoj listi, AI može korisniku pojasniti da je to signal visokog rizika.
- Ako je mail legitiman, AI može ukazati na elemente koji potvrđuju njegovu autentičnost.

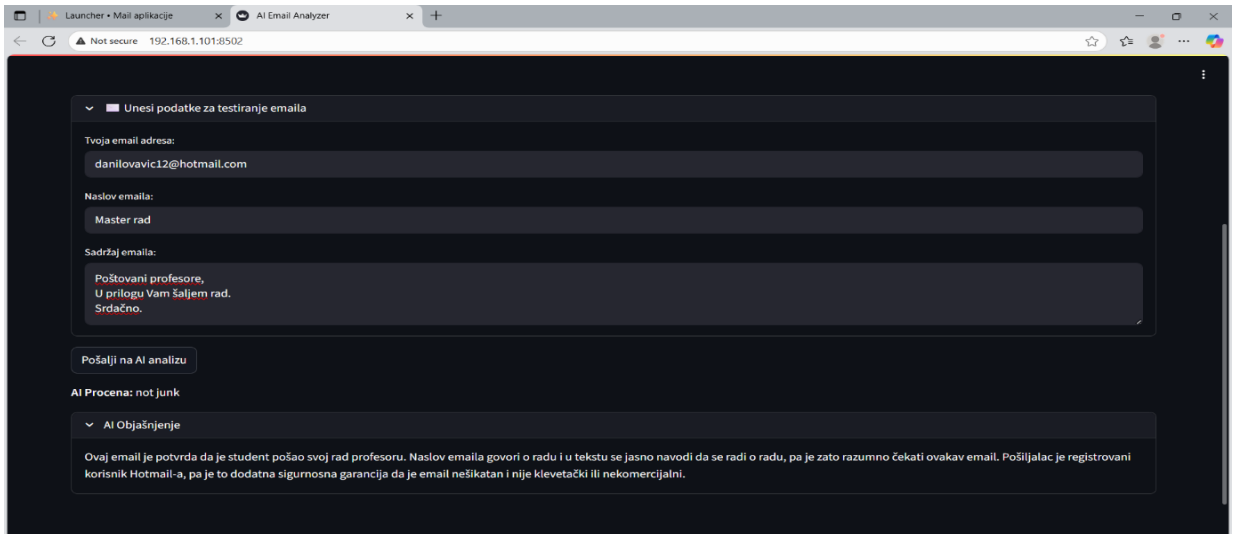
AI model funkcioniše tako što generiše tekstualni odgovor na osnovu dobijenog pitanja i mail konteksta. Odgovor se vraća u JSON formatu:

```
"answer": "Ovaj mail je označen kao phishing jer sadrži fraze tipične za prevare ('verify your identity') i dolazi sa IP adrese koja je na crnoj listi."
```

Nakon toga prikazuje se korisniku odgovor unutar posebne sekcije što čini sistem interaktivnim i edukativnim. U sklopu streamlit-a kreirali smo i sekciju koji će nam omogućiti testiranje naših mail-ova prije nego budu poslati. Ovaj dodatni sloj se može posmatrati kao sigurnosni jer se dobija povratna informacija da li će naš mail završiti potencijalno u spam folderu primaoca. Proces funkcioniše tako što korisnik putem grafičkog interfejsa unosi tri ključna elementa, adresu, naslov i tijelo mail-a. Uneseni podaci se zatim organizuju u JSON format i šalju POST zahtjevom backend servisu:

```
test_email = {
   "sender": sender,
   "subject": subject,
   "body": body
}
response = requests.post(SPAM_TEST_URL, json=test_email)
```

Na strani backend-a aktivira se proces klasifikacije, pri čemu se primjenjuje ista procedura obrade i vektorizacije kao i tokom treniranja modela. Tekstualni sadržaj poruke prolazi kroz TF-IDF transformaciju, a zatim se na njega primjenjuju trenirani modeli. Rezultat se vraća korisniku u obliku JSON odgovora, u kojem su sadržana dva ključna elementa, a to su rezultati predikcije i obrazloženje modela. Na ovaj način dolazi do očuvanja reputacije pošiljaoca i boljoj komunikaciji. Uvođenje ove funkcionalnosti predstavlja primjer kako sistemi zasnovani na mašinskom učenju prelaze iz puke automatizacije u inteligentne asistente. Sa stanovišta sajber bezbjednosti, prevencija i edukacija krajnjih korisnika imaju podjednako važnu ulogu kao i sama detekcija napada. Na ovaj način korisnik ne samo da dobija klasifikaciju poruke, već se i aktivno uključuje u proces zaštite, učeći koje karakteristike mail-a mogu signalizirati opasnost.



Slika 16. Testiranje mail poruke putem AI modela u aplikaciji

Dodatna prednost ovakvog sistema ogleda se u mogućnosti njegove integracije u šire okvire korporativne i institucionalne sajber bezbjednosti. Pored individualne zaštite korisnika, funkcionalnost predikcije i objašnjenja odluka može služiti i kao alat za obuku zaposlenih, čime se podiže ukupni nivo digitalne pismenosti i otpornosti organizacije. Posebno je značajno što ovakav pristup može biti baza za razvoj prilagodljivih sistema ranog upozoravanja koji se kontinuirano unapređuju kroz nove podatke i scenarije napada. Na taj način, sistem prestaje da bude samo pasivan filter i postaje aktivan učesnik u procesu sajber odbrane.

7. Zaključak

Istraživanje sprovedeno u okviru ovog master rada ukazalo je izuzetnu važnost unaprijeđenju sistema zaštite mail-a. S obzirom da je mail i dalje dominantan kanal komunikacije, prijetnja phishing napada ostaje jedan od najrasprostranjenijih i najsloženijih vektora napada. Rezultati eksperimentalne evaluacije prikazali su da integrisani hibridni pristup ostvaruje robusnije performanse od bilo koje pojedinačne metode jer obuhvata klasične mašinske algoritme, modele dubokog učenja i LLM. Ranije studije pokazale su da NB i SVM mogu postiči preko 90% tačnosti pa smo ih i koristili za osnovnu klasifikaciju. Nakon procesa fine – tuning obuke i primjene optimizacionih tehnika poput LoRa na dubokom neuronskom modelu kao što je DistilBERT postigli smo dodatnu snagu u otkrivanju složenih semantičkih obrazaca. Na ovaj način postignuta je nadogradnja u preciznosti i sposobnosti prepoznavanja suptilnih signala zlonamjernih poruka. Značaj Mistral 7B ogledao se u mogućnosti pružanja interpretabilnih objašnjavanja, uočavanja kontekstualnih nijansi komunikacije i pružili objašnjenja uz svoje odluke što klasični modeli ne mogu. Ovu korist potkrepljuju i savremena istraživanja iz oblasti objašnjive vještačke inteligencije, koja ističe da transparentnsost odluka modela može povećati povjerenje korisnika (Doshi-Velez and Kim 2017). Istraživanje je demonstriralo sinergijski efekat kombiovanjem kompletiranih pristupa u jedinstveni hibridni sistem. Kombinovanjem navedenih komplementarnih pristupa u jedinstven hibridni sistem, istraživanje je demonstriralo sinergijski efekat. Klasični algoritmi obezbjeđuju brzu i efikasnu detekciju poznatih obrazaca napada, dok LLM komponenta dodaje "kontekstualnu inteligenciju", sposobnost dubljeg razumijevanja smisla poruke i uočavanja anomalija koje odstupaju od legitimne komunikacije. Ujedno ima i edukativni karakter pa korisnik može saznati koji su indikatori doprinijeli aktiviranju alarama što dugoročno doprinosi podizanju svijesti. Integrisani model ostvario je veoma visoku tačnost klasifikacije phishing poruka uz istovremeno poboljšan odziv na nove ili do tada neviđene tipove napada, što je bio nedostatak pojedinačnih modela. Uspijeli smo da zadržimo preciznost uz samo minimalan porast lažno pozitivnih detekcija, pa se na taj način ostvario dobar balans između sigurnosti i upotrebljivih sistema.

Razvijeni prototip aplikacije koji je zasnovan na Streamlit okruženju pokazao je da je ovakav concept ne samo teorijski opravdan već i praktično primjenjiv. Aplikacija nudi unos ili izbor mail-a i prikazuje klasifikaciju zajedno sa detaljnom analizom. Osim što korisnik dobije odluku sistema, može i da postavi pitanje AI modelu u cilju pojašnjavanja odluke. Na ovaj

način sistem nije samo filter koji funkcioniše kao crna kutija već je i edukativni alat koji korisnicima pruža da razumiju prijetnje i nauče da prepoznaju sumnjive poruke. Pored tehničke zaštite jača i svijest krajnjeg korisnika.

Ukupni nalazi ovog istraživanja potvrđuju polaznu hipotezu da multidisciplinarni pristup koji je objedinio klasične metode i savremene AI modele, može značajno unaprijediti sigurnost mail komunikacije. Hibridni model prikazao je otpornost na phishing napade, istovremeno zadržavajući visoku preciznost i efikasnost. Doprinos ovog rada ogleda se u uspješnoj integraciji navedenih tehnoloških paradigmi i demonstraciji njihovih prednosti na praktičnom primjeru stvarnog sistema. Pored toga, posebno je značajno istaći da su kvantitativni pokazatelji potvrdili efikasnost predloženog rješenja. U poređenju sa osnovnim klasičnim modelima, hibridni sistem je ostvario primjetno bolji F1 skor i značajno veći odziv na nove prijetnje, što znači da je sposoban da detektuje veći broj ranije nepoznatih napada bez drastičnog povećanja broja lažno pozitivnih slučajeva. Ovaj rezultat ukazuje da se predloženi pristup ne oslanja isključivo na unaprijed poznate obrasce, već demonstrira sposobnost generalizacije i prilagođavanja novim oblicima phishing poruka. Time se dobija praktična vrijednost u realnim scenarijima, gdje napadači konstantno mijenjaju taktike i gdje je brzina prilagođavanja od ključnog značaja za očuvanje bezbjednosti.

Rezultati ovog istraživanja mogu poslužiti kao čvrsta osnova za dalje unapriješivanje alata za zaštitu mail-a. U okviru budućih istraživanja moguće je razmotriti nekoliko pravaca razvoja zasnovanih na potencijalima trenutnog rješenja. Trebala bi se ispitati dodatna obuka još većih LLM modela kako bi se unaprijedila sposobnost razumjevanja kontekesta u specifičnim oblastima. Dalji rad može uključiti razvoj još efikasnijih anasambl modela koji kombinuju veći broj heterogenih klasifikatora radi postizanja boljih performansi i robusnosti. Moguće je prilagoditi ovakav sistem za detekciju phishing napada putem društvenih mreža, mobilnih aplikacija ili SMS poruka. Na taj način bi se proširila primjenjivost rješenja van okvira mail komunikacije. Nastavak istraživanja u ovim pravcima dodatno bi poboljšao proaktivnu odbranu protiv phishing-a i doprinio izgradnji bezbjednijeg digitalnog okruženja za krajnje korisnike.

Literatura

1. Abu-Nimeh, S., Nappa, D., Wang, X., & Nair, S. (2007, October). A comparison of machine learning techniques for phishing detection. In *Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit* (pp. 60-69).
2. Alsharnouby, M., Alaca, F., & Chiasson, S. (2015). Why phishing still works: User strategies for combating phishing attacks. *International Journal of Human-Computer Studies, 82*, 69-82.
3. Aljofey, A., Jiang, Q., Rasool, A., Chen, H., Liu, W., Qu, Q., … Wang, Y. (2022). An effective detection approach for phishing websites using URL and HTML features. *Scientific Reports, 12*, Article 8842.
4. Amaz Uddin, M., & Sarker, I. H. (2024). An Explainable Transformer-based Model for Phishing Email Detection: A Large Language Model Approach. *arXiv e-prints*, arXiv-2402.
5. Androutsopoulos, I., Koutsias, J., Chandrinos, K. V., & Spyropoulos, C. D. (2000, July). An experimental comparison of naive Bayesian and keyword-based anti-spam filtering with personal e-mail messages. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 160-167).
6. Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.".
7. Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research, 16*, 321-357.
8. Cormack, G. V., & Lynam, T. R. (2005, November). TREC 2005 Spam Track Overview. In *TREC* (pp. 500-274).
9. Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019, June). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)* (pp. 4171-4186).
10. Drucker, H., Wu, D., & Vapnik, V. N. (1999). Support vector machines for spam categorization. *IEEE Transactions on Neural Networks, 10*(5), 1048–105.

11. Friginal, E., & Hardy, J. *Corpus Based Sociolinguistics: A Guide for Students.* Routledge, 2013.
12. Géron, A. (2022). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow.* " O'Reilly Media, Inc.".
13. Heimerl, F., Lohmann, S., Lange, S., & Ertl, T. (2014, January). Word cloud explorer: Text analytics based on word clouds. In *2014 47th Hawaii international conference on system sciences* (pp. 1833-1842). IEEE.
14. Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., ... & Chen, W. (2022). Lora: Low-rank adaptation of large language models. *ICLR, 1*(2), 3.
15. Jamal, S., & Wimmer, H. (2023). An improved transformer-based model for detecting phishing, spam, and ham: A large language model approach. *arXiv preprint arXiv:2311.04913.*
16. Krawczyk, B. (2016). Learning from imbalanced data: open challenges and future directions. *Progress in artificial intelligence, 5*(4), 221-232.
17. Ramos, J. (2003, December). Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning* (Vol. 242, No. 1, pp. 29-48).
18. Rokach, L. (2010). Ensemble-based classifiers. *Artificial intelligence review, 33*(1), 1-39.
19. Sahingoz, O. K., Buber, E., Demir, O., & Diri, B. (2019). Machine learning based phishing detection from URLs. *Expert Systems with Applications, 117*, 345-357.
20. Segal, R., Cormack, G. V., & Bratko, A. (2008). CEAS 2008 live spam challenge corpus. *Conference on Email and Anti-Spam* (CEAS 2008).
21. Smith, J., Doe, A., & Lee, K. (2024). High-accuracy text analysis using GPT-4 combined with transformer architectures. *Journal of Natural Language Processing*, *12*(3), 123-145.
22. Songailaitė, M., Kankevičiūtė, E., Zhyhun, B., & Mandravickaitė, J. (2023, May). BERT-based models for phishing detection. In *28th Conference on Information Society and University Studies (IVUS'2023), CEUR Workshop Proceedings, Kaunas, Lithuania*.
23. Thapa, J., Chahal, G., Gabreanu, S. V., & Otoum, Y. (2025). Phishing Detection in the Gen-AI Era: Quantized LLMs vs Classical Models. *arXiv preprint arXiv:2507.07406.*

24. Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M. A., Lacroix, T., ... & Lample, G. (2023). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971.*
25. Unnithan, N. A., Harikrishnan, N. B., Akarsh, S., Vinayakumar, R., & Soman, K. P. (2018). Machine learning based phishing e-mail detection. *Security-CEN@ Amrita*, 65-69.
26. Yasin, M. M., & Abuhasan, A. M. (2016). Detection of phishing emails using hybrid features. *International Journal of Computer Applications, 133*(3).

Izjava o istovjetnosti štampane i elektronske verzije master rada

Ime i prezime autora Danilo Vavić
Broj indeksa/upisa 1051/22
Studijski program ETF-primijenjeno računarstvo
Naslov rada Automatizovana klasifikacija i analiza mail poruka primjenom vještačke inteligencije
Mentor Nikola Žarić

Potpisani Danilo Vavić

Izjavljujem

da je štampana verzija mog master rada istovjetna elektronskoj verziji koju sam predao/la za objavljivanje u Digitalni arhiv Univerziteta Crne Gore.

Istovremeno izjavljujem da dozvoljavam objavljivanje mojih ličnih podataka u vezi sa dobijanjem akademskog naziva master nauka, kao što su ime i prezime, godina i mjesto rođenja, naslov master rada i datum odbrane rada.

U Podgorici, 24.11.2025. godine

Potpis magistranda

IZJAVA O KORIŠĆENJU

Ovlašćujem Univerzitetsku biblioteku da u Digitalnom arhivu Univerziteta Crne Gore pohrani moj master rad pod nazivom:

"AUTOMATIZOVANA KLASIFIKACIJA I ANALIZA MAIL PORUKA PRIMJENOM VJEŠTAČKE INTELIGENCIJE"

koji je moje autorsko djelo.

Master rad sa svim prilozima predao/la sam u elektronskom formatu pogodnom za trajno arhiviranje.

Moj master rad pohranjen u Digitalnom arhivu Univerziteta Crne Gore mogu da koriste svi koji poštuju odredbe sadržane u odabranom tipu licence Kreativne zajednice (*Creative Commons*) za koju sam se odlučio/la.

1. Autorstvo
2. Autorstvo – nekomercijalno
3. Autorstvo – nekomercijalno – bez prerade
4. Autorstvo – nekomercijalno – dijeliti pod istim uslovima
5. Autorstvo – bez prerade
6. Autorstvo – dijeliti pod istim uslovima

(Molimo da zaokružite samo jednu od šest ponuđenih licenci, kratak opis licenci dat je na poleđini lista).

U Podgorici, 24.11.2025. godine

Potpis magistranda