



УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ
НАУКА У НОВОМ САДУ




Марко Митошевић

**Додавање подршке за
библиотеку Keras 3 у радно
окружење TensorFlow Federated**


ЗАВРШНИ РАД

Основне академске студије

Нови Сад, 2025

	УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА 21000 НОВИ САД, Трг Доситеја Обрадовића 6	
	КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА	

Редни број, РБР:		
Идентификациони број, ИБР:		
Тип документације, ТД:		Монографска документација
Тип записа, ТЗ:		Текстуални штампани материјал
Врста рада, ВР:		Дипломски - бечелор рад
Аутор, АУ:		Марко Митошевић
Ментор, МН:		Др Игор Дејановић, редовни професор
Наслов рада, НР:		Додавање подршке за библиотеку Keras 3 у радно окружење TensorFlow Federated
Језик публикације, ЈП:		српски/ћирилица
Језик извода, ЈИ:		српски/енглески
Земља публиковања, ЗП:		Република Србија
Уже географско подручје, УГП:		Војводина
Година, ГО:		2025
Издавач, ИЗ:		Ауторски репринт
Место и адреса, МА:		Нови Сад, трг Доситеја Обрадовића 6
Физички опис рада, ФО: (поглавља/страница/ цитата/табела/слика/графика/прилога)		5/40/19/2/14/0/2
Научна област, НО:		Електротехничко и рачунарско инжењерство
Научна дисциплина, НД:		Примењене рачунарске науке и информатика
Предметна одредница/Кључне речи, ПО:		Keras, TensorFlow Federated, IDE
УДК		
Чува се, ЧУ:		У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, ВН:		
Извод, ИЗ:		<p>Примарна мотивација овог рада је унапређење алата за претрагу <i>Search Everywhere</i> (SE) у интегрисаним развојном окружењу (IDE) <i>IntelliJ</i>, помоћу модела машинског учења. Због захтева за приватношћу података корисника, потребна је примена федеративног учења (FL). Међутим, водеће FL окружење, TensorFlow Federated (TFF) има недостатак подршке за библиотеку Keras 3. Овај рад се бави анализом, дизајном и имплементацијом решења за додавање подршке за библиотеку Keras 3 у TFF. Циљ је постигнут имплементацијом паралелне подршке, чиме је очувана пуна компатибилност са постојећом Keras 2 архитектуром.</p>
Датум прихватања теме, ДП:		
Датум одбране, ДО:		01.01.2025
Чланови комисије, КО:	Председник:	Др Петар Петровић, ванредни професор
	Члан:	Др Марко Марковић, доцент
	Члан:	
	Члан, ментор:	Др Игор Дејановић, редовни професор
		Потпис ментора

	UNIVERSITY OF NOVI SAD • FACULTY OF TECHNICAL SCIENCES 21000 NOVI SAD, Trg Dositeja Obradovića 6	
	KEY WORDS DOCUMENTATION	
Accession number, ANO :		
Identification number, INO :		
Document type, DT :	Monographic publication	
Type of record, TR :	Textual printed material	
Contents code, CC :		
Author, AU :	Marko Mitosevic	
Mentor, MN :	Igor Dejanović, Phd., full professor	
Title, TI :	Adding support for the Keras 3 library to the TensorFlow Federated framework	
Language of text, LT :	Serbian	
Language of abstract, LA :	Serbian/English	
Country of publication, CP :	Republic of Serbia	
Locality of publication, LP :	Vojvodina	
Publication year, PY :	2025	
Publisher, PB :	Author's reprint	
Publication place, PP :	Novi Sad, Dositeja Obradovica sq. 6	
Physical description, PD : (chapters/pages/ref./tables/pictures/graphs/appendixes)	5/40/19/2/14/0/2	
Scientific field, SF :	Electrical and Computer Engineering	
Scientific discipline, SD :	Applied computer science and informatics	
Subject/Key words, S/KW :	Keras, TensorFlow Federated, IDE	
UC		
Holding data, HD :	The Library of Faculty of Technical Sciences, Novi Sad	
Note, N :		
Abstract, AB :	<p>The primary motivation for this paper is the improvement of the Search Everywhere (SE) tool within the IntelliJ Integrated Development Environment (IDE) using machine learning models. Due to user data privacy requirements, the application of Federated Learning (FL) is necessary. However, the leading FL framework, TensorFlow Federated (TFF), lacks support for the Keras 3 library. This paper addresses the analysis, design, and implementation of a solution for adding Keras 3 library support to TFF. The objective was achieved by implementing parallel support, thereby preserving full compatibility with the existing Keras 2 architecture.</p>	
Accepted by the Scientific Board on, ASB :		
Defended on, DE :	01.01.2025	
Defended Board, DB :	President:	Petar Petrović, Phd., assoc. professor
	Member:	Marko Marković, Phd., asist. professor
	Member:	
	Member, Mentor:	Igor Dejanović, Phd., full professor
		Menthor's sign



УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
21000 НОВИ САД, Трг Доситеја Обрадовића 6

ИЗЈАВА О НЕПОСТОЈАЊУ СУКОБА ИНТЕРЕСА

Изјављујем да нисам у сукобу интереса у односу ментор – кандидат и да нисам члан породице (супружник или ванбрачни партнер, родитељ или усвојитељ, дете или усвојеник), повезано лице (крвни сродник ментора/кандидата у правој линији, односно у побочној линији закључно са другим степеном сродства, као ни физичко лице које се према другим основама и околностима може оправдано сматрати интересно повезаним са ментором или кандидатом), односно да нисам зависан/на од ментора/кандидата, да не постоје околности које би могле да утичу на моју непристрасност, нити да стичем било какве користи или погодности за себе или друго лице било позитивним или негативним исходом, као и да немам приватни интерес који утиче, може да утиче или изгледа као да утиче на однос ментор-кандидат.

У Новом Саду, дана _____

Ментор

Кандидат

Садржај

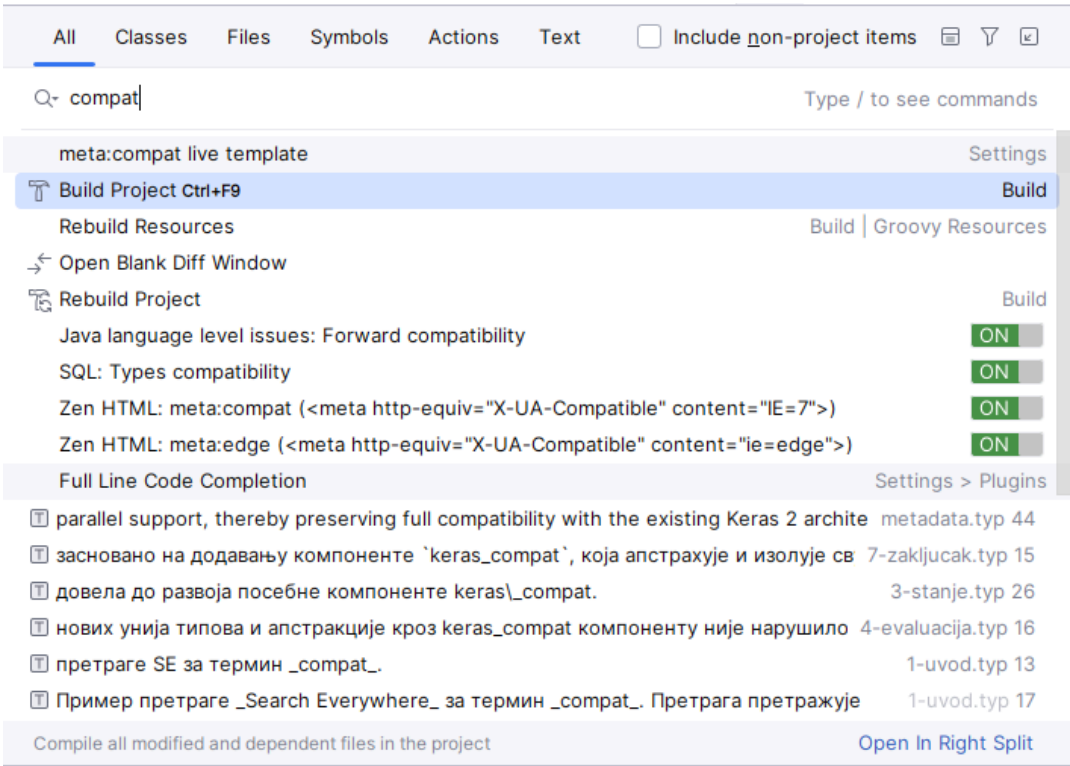
1	Увод	1
1.1	Мотивација	1
1.2	Циљеви рада	2
1.3	Организација рада	3
2	Теоријске основе	5
2.1	Федеративно учење	5
2.2	Оптимизована диференцијална приватност	6
2.3	Радно окружење <i>TensorFlow Federated</i> (TFF)	8
2.4	Библиотека <i>Keras</i>	9
3	Имплементација подршке за <i>Keras 3</i>	11
3.1	Анализа имплементације проблема компатибилности	11
3.2	Архитектура рефакторисања	12
3.3	Апстракција логике руковања моделом	13
3.4	Примена апстракције и уније типова	18
4	Евалуација решења	21
4.1	Верификација успешности решења	21
4.2	Предности решења	22
4.3	Дискусија о решењу	22
4.4	Утицај рефакторисања TFF екосистема	23
5	Закључак	25
5.1	Резиме рада и остварени циљеви	25
5.2	Кључни доприноси и значај рада	25
5.3	Ограничења и будући рад	25
	Списак слика	27
	Списак листинга	29
	Списак табела	31
	Списак коришћених скраћеница	33
	Списак коришћених појмова	35
	Биографија	37
	Литература	39

Глава 1

Увод

1.1 Мотивација

Модерна интегрисана развојна окружења (*Integrated Development Environment*, IDE) имају велики број функционалности које помажу програмерима да ефикасно рукују пројектима великог обима. Једна од кључних функционалности у IDE-овима, попут *IntelliJ IDEA*, је претрага *Search Everywhere* (SE), која кориснику на једном месту омогућава да претражи све функционалности окружења, као и све датотеке и њихов садржај унутар пројектне структуре [1]. На слици 1 приказан је пример претраге SE за термин *compat*.



Слика 1: Пример претраге *Search Everywhere* за термин *compat*. Претрага претражује функционалности IDE, као и пројектну структуру како ради проналажења свих појављивања термина претраге унутар пројекта.

Корисници често не знају тачно име функционалности коју желе да пронађу, што отежава претрагу. Како би се овај проблем ублажио, истражују се решења модела машинског учења (ML) која могу да предвиде име жељене функционалности. Коришћењем ових модела могуће је направити препоруку претраге, чиме би се олакшало коришћење функционалности SE. Препорука би била најефикаснија уколико би модел био трениран над претходним претрагама корисника и других корисника, као и над контекстом под којим је претрага позвана.

Тренирање модела над осетљивим корисничким подацима представља изазов у погледу приватности и поверљивости. Многе компаније морају чувати поверљивост свог кода и информационих система. Уредбе о заштити података корисника попут Опште уредбе о заштити података (*General Data Protection Directive*, GDPR), која је ступила на снагу у Европској Унији, забрањују обраду података без експлицитне сагласности корисника или адекватног правног основа [2]. Наведена ограничења онемогућавају директно слање података на централни сервер ради тренирања модела.

Постоје технике ML које имају способност да чувају поверљивост тренинг података. Најпознатије су федеративно учење (*Federated Learning*, FL) [3] и диференцијална приватност (*Differential Privacy*, ODP) [4]. Ове технике се базирају на дистрибутивном ML којим се гарантује поверљивост података. Мана код тренутне имплементације FL у радном окружењу *TensorFlow Federated* (TFF) је недостатак подршке за учитавање најновијих модела, који су потребни да тачно предвиде претрагу SE. Недостатак подршке постоји због имплементације TFF која се ослања на претходну верзију библиотеке *Keras*, *Keras 2*, која за разлику од наредне верзије библиотеке не подржава учитавање модерних модела [5]. Елиминисањем ове мане би се отворила могућност за шире коришћење FL у индустрији.

Циљ овог рада је додавање подршке за новије моделе у радно окружење отвореног кода *TensorFlow Federated* (TFF), које имплементира технику FL [5].

1.2 Циљеви рада

Главни циљ овог рада је да се имплементира функционална подршка за библиотеку *Keras 3* унутар радног окружења TFF. Да би се то постигло, потребно је испунити следеће циљеве:

- Анализирати постојећу архитектуру TFF-а и њену зависност од библиотеке *Keras 2*.
- Рефакторисати компоненте TFF-а тако да би се постигла компатибилност са верзијом 3 библиотеком *Keras*, уз очување компатибилности са верзијом 2.
- Имплементирати компоненту која омогућава јединствен начин руковања моделима и компонентама обе верзије библиотеке *Keras*.

- Имплементирати тестове за тестирање компатибилности библиотеке *Keras 3* са рефакторисаним компонентама TFF.
- Евалуирати функционалну исправност имплементиране подршке проласком свих тестова.
- Демонстрирати предности интеграције *Keras 3* коришћењем модела са знатно бољим перформансама (нпр. *Gemma 3*) у односу на *Keras 2* моделе (нпр. GPT-2).

1.3 Организација рада

Рад је организован у пет поглавља. Прво поглавље, Увод, дефинише мотивацију и циљеве рада, уводећи читаоца у проблематику приватности података у машинском учењу и циљ рада. Друго поглавље, Теоријске основе, описује концепт федеративног учења, архитектуру и примену радног окружења TFF, као и библиотеке *Keras*. Треће поглавље, Имплементација подршке за *Keras 3*, описује детаље рефакторисања кода и увођење нових компатибилних функционалности. Четврто поглавље, Резултати и дискусија, представља евалуацију решења и поређење перформанси нових и старих модела. Пето поглавље, Закључак, сумира постигнуте резултате и предлаже правце за даља унапређења.

Глава 2

Теоријске основе

2.1 Федеративно учење

Федеративно учење (*Federated Learning*, FL) је техника машинског учења која омогућава децентрализовано тренирање модела над поверљивим подацима. За разлику од традиционалног централизованог приступа, где се сви подаци прикупљају на једном серверу, FL гарантује да подаци никада не напуштају оригинални уређај. Овај приступ је кључан за очување поверљивости и усклађеност са регулативама за заштиту података [3].

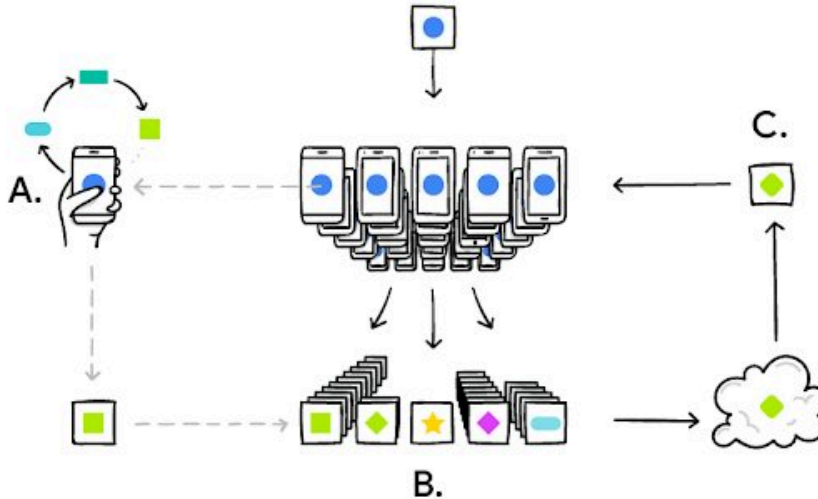
Систем федеративног учења функционише на принципу децентрализоване обуке. Сваки уређај (клијент) у систему поседује локалну копију дељеног модела. Локални модел се тренира искључиво коришћењем података који се налазе на том уређају. Након завршетка локалног тренинга, уређај не шаље своје приватне податке, већ само ажуриране тежине модела централном серверу.

Централни сервер има улогу агрегатора. Он прикупља ажуриране тежине од великог броја уређаја и агрегира резултате. Израчунати, побољшани глобални модел се затим шаље назад свим уређајима у систему, и циклус обуке се понавља. Побољшани глобални модел замењује локални модел на клијентским уређајима и постаје почетна тачка за следећу рунду тренирања. Процес се изводи итеративно, чиме се временом побољшавају перформансе дељеног модела, без употребе приватних података.

Један од кључних изазова и мера перформанси овог итеративног процеса јесте директна веза између величине модела и системских трошкова. Већи, комплекснији модели, попут модела *Gemma 3* [6], нуде значајно бољу тачност предвиђања, али доносе два практична оптерећења. Прво, захтевају веће рачунарске ресурсе на самом клијентском уређају (меморију и процесорску снагу) за извођење локалног тренинга. Друго, генеришу велика ажурирања тежина (која могу имати милијарде параметара), што ствара комуникационо уско грло (*communication bottleneck*) приликом слања ажурирања са хиљада или милиона уређаја на централни сервер.

Ефикасност федеративног учења не зависи само од алгорита агрегације, већ и од техника за компресију модела и смањење комуникационог оптерећења. Немогућност коришћења новијих, боље оптимизованих модела, попут модела *Llama 3* [7], који су дизајнирани да понуде бољи однос величине и перформанси, представља проблем који овај рад решава.

Потенцијалне области примене федеративног учења су индустрије које захтевају учење модела над поверљивим скуповима података великог броја корисника, као што су медицина, роботика и софтверско инжењерство. Најутицајнија примена FL-а над великим скупом података је предвиђање корисничког уноса на Gboard Android тастатури. На слици 2 приказан је примена FL за предвиђање корисничког уноса на Gboard Android тастатури [8].



Слика 2: Примена FL за предвиђање корисничког уноса на Gboard Android тастатури. Мобилни уређај локално тренира над претходним корисничким уносима и ажуриране тежине шаље централном серверу (A). Централни сервер агрегира примљене тежине (B) и шаље свим уређајима побољшан дељени модел (C) [8].

Кључна предност Федеративног учења је гарантована поверљивост података, јер подаци остају искључиво на оригиналним уређајима, што омогућава децентрализовано тренирање. Ова техника, заједно са оптимизованом диференцијалном приватношћу, базира се на дистрибутивном машинском учењу којим се ефикасно чува поверљивост тренинг података.

2.2 Оптимизована диференцијална приватност

Диференцијална приватност (*Differential Privacy*, DP) је математички дефинисан оквир који квантификује и гарантује приватност појединца у оквиру базе података. Основна идеја је увођење контролисаног шума у процес обраде података, тако да присуство или одсуство записа једног појединца у скупу података не може значајно утицати на коначни излаз анализе или модела. Овај концепт пружа гаранцију да

нападач не може са сигурношћу закључити да ли је одређени појединац био део скупа података [9].

Математичка дефиниција DP се најчешће изражава параметрима ϵ (епсилон) и δ (делта). Алгоритам A је (ϵ, δ) диференцијално приватан ако за свака два скупа података D и D' , која се разликују за само један узорак и за сваки излазни скуп резултата S важи следећа неједнакост [10]:

$$P[A(D) \in S] \leq e^\epsilon * P[A(D') \in S] + \delta$$

, где је P вероватноћа. Загарантовано је да се вероватноћа добијања било ког излаза не може променити за фактор већи од e^ϵ ако се један узорак дода или уклони.

Параметар ϵ одређује горњу границу промене вероватноће излаза приликом укључивања или изостављања једног узорака података. Нижа вредност ϵ означава бољу приватност, али потенцијално и мању тачност модела. За ϵ једнак нули излази су индентични и добија се загарантована приватност, што је у пракси немогуће достићи.

Параметар δ представља вероватноћу да горња граница ϵ неће бити задовољена. У идеалном теоријском случају δ тежи нули, чиме се добија диференцијална приватност која зависи само од параметра ϵ , док се у практичним применама поставља на малу вредност (нпр. 10^{-5}), која би требало да буде мања од инверзне вредности величине скупа података ($\frac{1}{\text{величина скупа података}}$), чиме се смањује ризик угрожавања приватности појединог узорака у скупу података.

Оптимизована диференцијална приватност (*Optimized Differential Privacy*, ODP) представља варијанту примене DP, која је специјализована за контекст машинског учења и великих, дистрибуираних система. Главни изазов код стандардне DP је постизање оптималног компромиса између приватности и тачности модела. ODP користи методе попут клиповања градијента (*Gradient Clipping*) [11] пре додавања шума, како би се смањио потребан ниво шума и одржала тачност модела, уз истовремено задовољавање дефинисане ϵ границе приватности.

У контексту дистрибутивног машинског учења, DP се може применити на два начина [12]:

- Локална диференцијална приватност (*Local Differential Privacy*, LDP): Шум се додаје директно на сирове податке или ажурирања модела на самом уређају (клијенту) пре слања података на сервер, чиме се добија заштита и од злонамерног централног сервера.
- Централна диференцијална приватност (*Central Differential Privacy*, CDP): Шум се додаје на централном серверу, након што су прикупљени сви доприноси клијената. Овај модел претпоставља да је сервер поуздан. Сервер агрегира све прикупљене

доприносе и додаје шум на финални, агрегирани резултат пре него што га примени на глобални модел или га учини јавно доступним, чиме се штити приватност скупа података као целине, али се не штити приватност појединачних клијената од самог сервера.

Технике FL и DP су комплементарне. FL пружа технички механизам за децентрализацију тренинга и спречава прикупљање сирових података на серверу, док DP/ODP пружа математичку гаранцију да чак ни ажурирања модела не могу открити податке појединаца. У моделу где је централни сервер непоуздан, што је најчешћи сценарио када се штити приватност корисника, примењује се локална диференцијална приватност (LDP). У овом приступу, сваки клијент примењује ODP локално, на сопственом уређају, пре слања података серверу. Клијент прво израчуна ажуриране тежине, затим клипује градијенте чиме ограничава њихову осетљивост [11] и на крају на њих додаје насумични шум. Тек тада се ове заштићене тежине шаљу серверу.

Централни сервер, који врши агрегацију, никада не види осетљиве, оригиналне тежине појединачних клијената, већ искључиво њихове заштићене верзије. На овај начин се спречава да сервер може закључити било какве информације о појединачним тренинг подацима. Ова комбинација је неопходна за изградњу модерних система машинског учења који су у потпуности усклађени са регулативама за заштиту података.

2.3 Радно окружење *TensorFlow Federated* (TFF)

TensorFlow Federated (TFF) је радно окружење отвореног кода које је развила компанија *Google* са циљем да олакша истраживање и примену FL у реалним окружењима [5]. Поред имплементације постојећих FL алгоритама, радно окружење TFF пружа и модуларни оквир за истраживање и дизајн нових дистрибуираних алгоритама. Главна предност је подршка за симулирање FL алгоритама користећи *TensorFlow* (TF) моделе над симулираним уређајима.

Архитектура TFF-а је подељена на два главна слоја [5]:

- Федеративни модел API (*Federated Model API*): слој намењен истраживачима и инжењерима машинског учења. Пружа скуп модула високог нивоа који омогућавају кориснику да постојеће моделе изграђене у TF-у, укључујући и *Keras* моделе, лако прилагоди за FL окружење. Главна функција овог слоја је аутоматско генерисање FL алгоритама попут федеративног просека (*FedAvg*) из стандардног модела машинског учења.
- Федеративни основни API (*Federated Core API*): темељ TFF-а, намењен за имплементацију нових FL алгоритама. Федеративни основни API је декларативни језик који омогућава експлицитно дефинисање дистрибуираних рачунарских операција. Овај језик је дизајниран за рад са федеративним типовима података, као што су подаци на клијентима (CLIENTS) и подаци на серверу (SERVER). Све комуника-

ционе операције, као што су агрегација и дифузија (слање модела), дефинишу се експлицитно, омогућавајући потпуну контролу над током података и комуникацијом ефикасношћу.

Кључна карактеристика TFF-а је његов систем федеративних типова. За разлику од библиотеке *TensorFlow* која ради само са тензорима (низовима података), TFF уводи концепт локације података. На пример, подаци који се налазе на свим клијентима дефинисани су као {тип}@CLIENTS, док су подаци који се налазе на централном серверу дефинисани као {тип}@SERVER. Јасно раздвајање локације података осигурава да се подаци не могу случајно преместити са приватне локације на централни сервер, чиме се пружа основна гаранција приватности.

Радно окружење TFF-а нуди подршку за учитавање *Keras* модела због њихове једноставности и популарности. TFF користи функцију `tff.learning.from_keras_model` за претварање *Keras* модела у `tff.learning.Model` интерфејс. Овај интерфејс омогућава коришћење традиционалног централизованог тренирања са дистрибуираним FL протоколима. Претворени модел се може користити за креирање комплетног федеративног итеративног процеса, који дефинише иницијализацију (`initialize`) и један корак тренирања (`next`) у FL циклусу.

Механизам конверзије и адаптације *Keras* модела је кључан за додавање подршке за верзију 3 библиотеке *Keras*. Постојећа архитектура `from_keras_model` је чврсто везана за архитектуру *Keras* 2 библиотеке. Због тога је за подршку новој верзији библиотеке *Keras* неопходно рефакторисање компоненти TFF-а које управљају стањем модела, оптимизаторима и метрикама, како би TFF могао да искористи предности новијих модела за машинско учење.

2.4 Библиотека *Keras*

Keras је библиотека отвореног кода за машинско учење која служи као интерфејс високог нивоа, омогућавајући једноставно коришћење сложених модела. Примарна сврха је једноставно коришћење популарних серверских радних окружења (*backend*) за машинско учење, као што су *TensorFlow*, *JAX* и *PyTorch*. *Keras* пружа конзистентан и једноставан апликативни програмски интерфејс (*Application Programming Interface*, API), који апстрахује сложеност *backend* имплементације, што омогућава истраживачима и инжењерима да се фокусирају на дизајн модела [13]. Такође, *Keras* подржава учитавање и тренирање модела са екстерних платформи, попут платформе *Kaggle*-а [14].

Док је верзија 2 библиотеке *Keras* била чврсто везана за *TensorFlow* као свој примарни *backend*, верзија 3 библиотеке *Keras* донела је темељну архитектонску промену. Најновија верзија имплементације библиотеке *Keras* је у потпуности редизајнирана да постане библиотека која подржава више *backend* окружења (*multi-backend*), чиме

се пружа флексибилност у избору најбољег окружења за специфичан хардвер или задатак. Због ових промена, *Keras 3* је морао да редефинише како се интерно рукује компонентама попут оптимизатора и метрика.

Главни значај библиотеке *Keras 3* за федеративно учење и овај рад лежи у његовим побољшањима перформанси и скалабилности. *Keras 3* је омогућио подршку за учитавање и тренирање модела са више милијарди параметара, као што је модел *Llama 3* [7]. Нови модели имају знатно боље перформансе у поређењу са претходним моделима са мањим бројем параметара. Немогућност коришћења ових напредних модела у TFF-у била је главна мотивација за рефакторисање радног окружења.

Архитектонска промена на *multi-backend* у библиотеци *Keras 3* има директан утицај на имплементацију TFF-а јер је библиотека *Keras 3* увела *stateless* парадигму за кључне компоненте. *Stateless* парадигма представља систем који не чува податке о претходним интеракцијама са корисником, док *statefull* парадигма чува претходне интеракције са корисником. У библиотеци *Keras 3*, оптимизатори више нису директно *statefull*, већ користе методе попут *stateless_apply*. Ово представља проблем код FL, јер TFF мора експлицитно да рукује променљивим стањима оптимизатора (*optimizer slots*) приликом агрегације и дистрибуције.

Слично променама у оптимизаторима, и метрике (*Metrics*) су прешле на *stateless* приступ (нпр. функција *stateless_update_state*), што је променило начин на који TFF издваја и агрегира нефинализоване метрике са клијената. Постојећа функција *from_keras_model* у TFF-у ослањала се на имплементацију *Keras 2* модела за трансформацију модела, које су у библиотеци *Keras 3* комплетно промењене. Функција би морала бити модификована, да би подржала нове структуре и понашања нове верзије библиотеке *Keras*.

Тренутна верзија TFF-а подржава само *Keras 2*, што значајно ограничава коришћење новијих, напреднијих модела. Додавањем подршке за *Keras 3*, омогућило би се тренирање над најновијим моделима, чиме би се знатно побољшало предвиђање претраге (SE) унутар IDE-а, што је крајњи циљ примене федеративног учења у овом домену.

Глава 3

Имплементација подршке за *Keras* 3

3.1 Анализа имплементације проблема компатибилности

Архитектонска промена библиотеке *Keras* из верзије 2 у верзију 3, описана у поглављу Библиотека *Keras*, је разлог проблема компатибилности библиотеке унутар радног окружења TFF. Функција `tff.learning.from_keras_model`, била је дизајнирана да очекује и користи унутрашње API позиве специфичне за *Keras* 2 (нпр. за директно издавање *statefull* променљивих оптимизатора). Пошто *Keras* 3 не подржава ове позиве, директна интеграција је постала немогућа.

Анализа TFF кода показала је да би потпуна миграција целог радног окружења TFF-а на *Keras* 3 архитектуру била претерано сложена и ризиковала би нарушавање функционалности постојећег, верификованог кода који се ослања на библиотеку *Keras* 2. Стога је донета одлука да се имплементира паралелна подршка за обе верзије библиотеке *Keras*. Овај приступ је захтевао да се TFF функције модификују тако да могу да рукују и са *Keras* 2 и са *Keras* 3 компонентама истовремено.

Паралелна подршка за обе верзије библиотеке *Keras* подразумева два примарна техничка захтева који су формирали оквир рефакторисања:

- Подршка за унију типова: функције у TFF-у морају бити у стању да прихвате објекте који су или *Keras* 2 или *Keras* 3 типови. Ово је решено имплементацијом уније типова за улазне параметре модела, метрика и функција грешке.
- Апстракција логике руковања моделом: креирање засебног механизма који би, на основу верзије *Keras* објекта, вратио одговарајуће променљиве (нпр. из *stateful Keras* 2 или *stateless Keras* 3 API-ја). Ова потреба је директно довела до развоја посебне компоненте `keras_compat`.

Рефакторисање се фокусирало на додавање логике компатибилности у TFF модуле *models*, *optimizers*, *metrics* и *algorithms*, омогућавајући рад са обе верзије библиотеке *Keras*, чиме је изазов миграције претворен у изазов апстракције.

3.2 Архитектура рефакторисања

Већина рефакторисаног кода се налази унутар сегмента радног окружења TFF који је имплементиран у програмском језику *Python* [15]. Унутар *Python* имплементације се налази секција *learning* у којој се налази код одговоран за алгоритме за федеративно учење, као и модули *models*, *optimizers*, *metrics*, и *algorithms* који рукују понашањем модела током обучавања [5].

Модул *models* представља најважнију тачку за интеграцију *Keras* модела у TFF. Његова примарна улога енкапсулација логики *Keras* модела у TFF-ов интерфејс `tff.learning.Model`. TFF интерфејс дефинише потребне атрибуте и методе које су кључне за FL процес, укључујући: обучиве променљиве (*trainable variables*), необучиве променљиве (*non-trainable variables*), променљиве стања модела (*model state variables*), методу `forward_pass` за прослеђивање података кроз модел и методу `report_local_unfinalized_metrics` за прикупљање локалних статистика [5].

Модул *optimizers* садржи логику неопходну за дефинисање и управљање оптимизаторима клијената (*client optimizers*). У федеративном учењу, оптимизатори имају клијентску и серверску улогу. На клијентској страни се користе за израчунавање локалних ажурирања модела на основу локалних података. Њихово стање, које укључује променљиве попут импулса (*momentum*) или адаптивних стопа учења (*adaptive learning rates*) мора бити укључено у стање сервера (*server state*). Овај модул обезбеђује да TFF може исправно да серијализује и десеријализује стање оптимизатора приликом дистрибуције и агрегације, омогућавајући континуирано учење током FL рунди [5]. Промене у понашању *Keras* 3 оптимизатора директно утичу на то како овај модул приступа њиховим променљивим стањима.

Модул *metrics* је одговоран за израчунавање, прикупљање и агрегацију статистичких мера перформанси модела, као што су тачност (*accuracy*) и губитак (*loss*) током процеса федеративног учења. У радном окружењу TFF, метрике се деле на две фазе: нефинализоване метрике и финализоване метрике. Клијенти израчунавају нефинализоване метрике (сирове вредности, као што су укупна сума губитака и број обрађених примера) и враћају их серверу. Овај модул садржи логику за агрегацију нефинализованих метрика преко свих клијената, као и логику за њихову финализацију (нпр. израчунавање просечне вредности) [5]. Прелазак *Keras* 3 метрика на *stateless* парадигму је захтевао рефакторисање модула, како би се обезбедило исправно извлачење и враћање нефинализованих тензора неопходних за централизовану агрегацију на серверу.

Модул *algorithms* садржи имплементације самих алгоритама федеративног учења и федеративне евалуације. Модул делује као координатор, користећи функционалности дефинисане у модулима *models*, *optimizers* и *metrics* за креирање комплетног итеративног процеса (*Iterative Process*). Итеративни процес је основна структура у TFF-у,

која дефинише стање сервера (*server state*) и следећи корак (*next step*) FL рунде. Модул *algorithms* је одговоран за имплементацију специфичних FL алгоритама, као што је федеративни просек (FedAvg) [5]. Због тога је рефакторисање овог модула било неопходно како би се осигурало да алгоритми функционишу са новим структурама података и функционалностима добијеним из рефакторисаних *models*, *optimizers* и *metrics* компоненти, обезбеђујући да је целокупни FL процес компатибилан са Keras 3 моделима.

3.3 Апстракција логики руковања моделом

Апстракција логики руковањем моделом је неопходна за несметано руковање различитим верзијама библиотеке Keras. Будући да постоје измене у руковању својстава модела између верзија библиотеке, потребно је увести компоненту која ће на основу библиотеке прикладно руковати са имплементацијом модела. Креирана је компонента `keras_compat` која садржи функције наведене у табели 1.

Табела 1: Функције компоненте `keras_compat` за апстракцију руковања моделом

Назив функције	Сврха функције
<code>is_compiled</code>	Проверава да ли је модел претходно компајлиран.
<code>keras_dtype_to_tf</code>	Претвара Keras типове података у TensorFlow типове података.
<code>is_keras3</code>	Проверава да ли објекат потиче из имплементације библиотеке Keras 3.
<code>get_optimizer_variables</code>	Добавља променљиве асоциране са оптимизатором (<i>optimizer</i>).
<code>ref</code>	Враћа референцу на променљиву која може бити Keras или TensorFlow типа.
<code>get_variable</code> и <code>get_variables</code>	Добавља променљиве Keras или TensorFlow модела.
<code>clone_model</code>	Враћа копију прослеђеног модела.
<code>int_shape</code>	Враћа облик модела као тип <i>int</i> .

3.3.1 Функција `_is_compiled`

Функција `_is_compiled` служи да провери да ли је прослеђени модел компајлиран, што је неопходан предуслов за започињање процеса машинског учења. Због разлика у интерним API позивима, функција мора експлицитно да разликује верзије, где се за објекте типа `tf.keras.Model` (имплементација Keras 2) провера врши преко приватног

атрибута `model._is_compiled`, док се за објекте типа `keras.Model` (имплементација *Keras* 3) користи директан атрибут `model.compiled`. Уколико је модел различитог типа од *Keras* типова модела, функција враћа грешку. Функција осигурава да радно окружење TFF увек може да потврди спремност модела за рад, без обзира на верзију *Keras* библиотеке. На листрингу 1 приказан је код функције.

```
def is_compiled(model: Model):
    if isinstance(model, tf_keras.Model):
        return model._is_compiled
    elif isinstance(model, keras.Model):
        return model.compiled
    else:
        raise TypeError(
            'Expected an instance of a tf_keras.Model, or a '
            'keras.Model; found a model of type '
            f'{type(model)}'
        )
```

Листинг 1: Функција `_is_compiled` из компоненте `keras_compat`

3.3.2 Функција `keras_dtype_to_tf`

Функција `keras_dtype_to_tf` осигурава конзистентности типова података (*data types*) у радном окружењу TFF. TFF је изграђен на библиотеци *TensorFlow*, која захтева специфичне `tf.dtype` објекте. Пошто *Keras* може да излаже типове података као једноставне `string` репрезентације (нпр. `float32`), функција мапира `string` вредности у њихове одговарајуће `tf.dtype` објекте. Поред стандардних типова, функција такође подржава и новије типове података уведене у библиотеци *TensorFlow*, као што су `float8_e4m3fn` и `float8_e5m2`, осигуравајући компатибилност са хардверски оптимизованим типовима. На листрингу 2 приказан је код функције.


```
def keras_dtype_to_tf(dtype_str):
    if not isinstance(dtype_str, str):
        return dtype_str
    return {
        "float16": tf.float16,
        "float32": tf.float32,
        "float64": tf.float64,
        "uint8": tf.uint8,
        "uint16": tf.uint16,
        "uint32": tf.uint32,
        "uint64": tf.uint64,
        "int8": tf.int8,
        "int16": tf.int16,
        "int32": tf.int32,
        "int64": tf.int64,
        "bfloat16": tf.bfloat16,
        "bool": tf.bool,
        "string": tf.string,
        "float8_e4m3fn": tf.dtypes.experimental.float8_e4m3fn,
        "float8_e5m2": tf.dtypes.experimental.float8_e5m2
    }.get(dtype_str, tf.float32)
```

Листинг 2: Функција `keras_dtype_to_tf` из компоненте `keras_compat`

3.3.3 Функција `is_keras3`

Функција `is_keras3` утврђује да ли прослеђени објекат (нпр. модел, променљива, метрика) потиче из *Keras 3* библиотеке. Провера се врши коришћењем `isinstance` методе уграђене у програмски језик *Python* над унијом различитих *Keras 3* класа као што су `keras.Model`, `keras.Variable`, `keras.Metric`, `keras.Layer`. Провера омогућава осталим функцијама компоненте да изаберу исправан API позив. На листрингу 3 приказан је код функције.

```
def is_keras3(obj: object):
    return isinstance(obj, (keras.Model, keras.Variable, keras.Metric,
        keras.Layer, keras.Loss, keras.Optimizer, keras_common.KerasVariable))
```

Листинг 3: Функција `is_keras3` из компоненте `keras_compat`

3.3.4 Функција `get_optimizer_variables`

Функција `get_optimizer_variables` решава проблем некомпатибилности у приступу променљивама оптимизатора између различитих верзија библиотеке *Keras*. У верзији 2, где су оптимизатори често имплементирани као `OptimizerV2` или сличне класе, атрибут `optimizer.variables` је функција (*callable*), која се мора позвати ради добијања листа променљивих. Насупрот томе, у верзији 3, `optimizer.variables` је директни атрибут. Функција апстрахује разлику и ако је `variables` функција, она је позива, у супротном, враћа атрибут директно, чиме се осигурава пренос стања оптимизатора

током FL рунди. Уколико прослеђени оптимизатор не припада ни једној од имплементација библиотеке *Keras*, функција враћа грешку. На листрингу 4 приказан је код функције.

```
def get_optimizer_variables(optimizer: Union[tf_keras.optimizers.Optimizer,
keras.optimizers.Optimizer, optimizer_v2.OptimizerV2]):
    if isinstance(optimizer, (tf_keras.optimizers.Optimizer,
keras.optimizers.Optimizer, optimizer_v2.OptimizerV2)):
        if callable(optimizer.variables):
            return optimizer.variables()
        return optimizer.variables
    else:
        raise TypeError(
            'Expected an instance of a tf_keras.optimizers.Optimizer, or a '
            'keras.optimizers.Optimizer; found an optimizer of type '
            f'{type(optimizer)}'
        )
```

Листинг 4: Функција `get_optimizer_variables` из компоненте `keras_compat`

3.3.5 Функција `ref`

Функција `ref` служи за добијање јединствене референце (ID) за променљиву, што је важно за интерно праћење стања у радном окружењу TFF. За `tf.Variable` (који се типично користи у имплементацији *Keras 2*), функција користи `variable.ref()` да би добила *TensorFlow* референцу. Међутим, за имплементацију *Keras 3* променљивих (`keras.Variable` или `keras_common.KerasVariable`), које су апстрахованије, функција враћа *Python* ID објекта користећи `id(variable)`. Уколико прослеђена променљива не припада ни једној од имплементација библиотеке *Keras*, функција враћа грешку. Овај механизам омогућава TFF-у да поуздано идентификује и мапира променљиве без обзира на њихов основни тип. На листрингу 5 приказан је код функције.

```
def ref(variable: Union[tf.Variable, keras.Variable,
keras_common.KerasVariable]):
    if isinstance(variable, tf.Variable):
        return variable.ref()
    elif isinstance(variable, (keras.Variable, keras_common.KerasVariable)):
        return id(variable)
    else:
        raise TypeError(f'Expected keras.Variable,
keras.backend.common.KerasVariable or tf.Variable, but got
{type(variable)}')
```

Листинг 5: Функција `ref` из компоненте `keras_compat`

3.3.6 Функције `get_variable` и `get_variables`

Функције `get_variable` и `get_variables` служе за издвајање стварне вредности променљиве. У библиотеци *Keras 3*, променљиве су представљене класама које имају атрибут `variable.value`, који садржи сам тензор. Функција `get_variable` проверава да ли је у питању имплементација *Keras 3* променљиве и, уколико јесте, враћа њену вредност. У супротном, враћа саму променљиву (што важи за `tf.Variable` у имплементацији *Keras 2*). Функција `get_variables` је помоћна функција која примењује `get_variable` на колекцију променљивих (листу или торку), обезбеђујући да радно окружење TFF увек рукује само са сировим тензорским вредностима. На листрингу 6 приказан је код функције `get_variable`, док је на листингу 7 приказан код функције `get_variables`.

```
def get_variable(variable: Union[tf.Variable, keras.Variable,
keras_common.KerasVariable]):
    if isinstance(variable, (keras.Variable, keras_common.KerasVariable)):
        return variable.value
    else:
        return variable
```

Листинг 6: Функција `get_variable` из компоненте `keras_compat`

```
def get_variables(variables: Union[list, tuple]):
    return [get_variable(var) for var in variables]
```

Листинг 7: Функција `get_variables` из компоненте `keras_compat`

3.3.7 Функције `clone_model`

Функција `clone_model` је одговорна за стварање копије модела са новим, неиницијализованим тежинама, што је неопходно у FL окружењу (нпр. приликом иницијализације клијентских модела). Будући да *Keras 3* и *Keras 2* имају засебне модуле за клонирање модела (`keras.models.clone_model` и `tf_keras.models.clone_model`), функција користи функцију `is_keras3` да одреди коју верзију функције за клонирање треба позвати, чиме се осигурава да радно окружење TFF поуздано клонира модел за сваког клијента, одржавајући исту архитектуру уз нове тежине. На листрингу 8 приказан је код функције `clone_model`.

```
def clone_model(model: Union[tf_keras.Model, keras.Model]):
    if is_keras3(model):
        return keras.models.clone_model(model)
    else:
        return tf_keras.models.clone_model(model)
```

Листинг 8: Функција `clone_model` из компоненте `keras_compat`

3.3.8 Функција `int_shape`

Функција `int_shape` враћа облик (`shape`) прослеђеног тензора или променљиве као торку целих бројева. Функција је стандардна помоћна функција која се користи у нижим слојевима машинског учења за ефикасно руковање димензијама, обезбеђујући да се димензије, које могу бити враћене као *TensorFlow* објекти, увек буду претворене у стандардни имплементацију торке (`tuple`) у програмском језику *Python*, чиме се поједностављује логика за проверу типова унутар радног окружења TFF. На листрингу 9 приказан је код функције `int_shape`.

```
def int_shape(x):
    try:
        shape = x.shape
        if not isinstance(shape, tuple):
            shape = tuple(shape.as_list())
        return shape
    except ValueError:
        return None
```

Листинг 9: Функција `int_shape` из компоненте `keras_compat`

3.4 Примена апстракције и уније типова

Постојећа функција `from_keras_model` у компоненти *models* претвара *Keras* модел у TFF модел. Да би истовремено подржала обе верзије библиотеке *Keras* употребљена је унија типова података. На листингу 10 приказан је исечак функције, који приказује имплементацију унију типова. Рефакторисан код је обележен знаковима на почетку реда, где је знаком минус приказан код пре рефакторисања, док је знаком плус приказан код додат у рефакторисању.

```
+ Model = Union[tf_keras.Model, keras.Model]
def from_keras_model(
    - keras_model: tf_keras.Model,
    + keras_model: Model,
```

Листинг 10: Исечак функције `from_keras_model`, који приказује имплементацију уније типова.

Поред примене уније типова примењена је и апстракција преко замене директних позива са позивом помоћних функција из додате компоненте `keras_compat`. На листингу 10 приказан је исечак функције `functional_model_from_keras`, који приказује додавање апстракције за добављање променљивих модела преко компоненте `keras_compat`. Рефакторисан код је обележен знаковима на почетку реда, где је знаком минус приказан код пре рефакторисања, док је знаком плус приказан код додат у рефакторисању.

```
trainable_variables = tuple(
    - v for v in captured_variables if v in cloned_model.trainable_variables
    + v for v in captured_variables if keras_compat.get_variable(v) in
keras_compat.get_variables(cloned_model.trainable_variables)
)
```

Листинг 11: Исечак функције `functional_model_from_keras`, који приказује додавање апстракције за добављање променљивих модела преко компоненте `keras_compat`.

Глава 4

Евалуација решења

4.1 Верификација успешности решења

Успешност имплементираног решења мери се његовом способношћу да обезбеди истовремену функционалност за обе верзије библиотеке *Keras* (2 и 3) унутар радног окружења TFF, уз очување стабилности постојећег кода.

За потребе функционалне верификације, коришћена је *Bazel* инфраструктура за изградњу и тестирање, која је уграђена у TFF пројекту [16]. *Bazel* омогућува да се тестови покрећу конзистентно у окружењу које симулира дистрибуирану архитектуру радног окружења TFF. Успешна верификација је захтевала да се број јединичних тестова повећа, тако што је свака постојећа *Keras* 2 тестна класа прилагођена да тестира и *Keras* 3 моделе, чиме је осигурана покривеност кода тестовима. Функционална верификација је спроведена на два нивоа:

- Потврда интегритета постојећег кода: Након рефакторисања, сви постојећи јединични тестови у TFF-у, који су се ослањали на *Keras* 2 имплементације су успешно извршени, чиме је доказало да увођење нових унија типова и апстракције кроз *keras_compat* компоненту није нарушило интегритет верификоване логике.
- Верификација *Keras* 3 функционалности: Имплементацијом нових јединичних тестова, доказано је да TFF алгоритми успешно иницијализују, обучавају и евалуирају моделе изграђене коришћењем *Keras* 3 API-ја.

Код решења се налази на *Github* репозиторијуму <https://github.com/markomitos/tensorflow-federated/pull/1>. Тестови се могу покренути командом у листингу 12, где *putanja* представља путању до жељене *Bazel* целине. *Bazel* целине у пројекту су одређене са *Bazel* BUILD датотекама, где свака BUILD датотека означава једну целину. На пример, за тестирање *metrics* секције радног окружења TFF, путања би била “//tensorflow_federated/python/learning/metrics”. Број тестова унутар *learning* директоријама пре рефакторисања је био 140, док је након реакторисања порастао на 279 тестова.

`bazel test putanja`

Листинг 12: Команда за покретање тестова у *Bazel* инфраструктури, где *putanja* представља путању до жељене *Bazel* целине

4.2 Предности решења

Постигнутим резултатом омогућено је коришћење најновијих модела и функционалности које *Keras 3* нуди, као што је подршка за више *backend* окружења, што отвара пут за будућа побољшања у радном окружењу TFF у погледу перформанси и компатибилности. Један од модела са високим перформансама који је могуће учитати због подршке за библиотеку *Keras 3* је *Gemma 3* [6]. У табели 2 налази се поређење перформансе модела *Gemma 3* у односу на *Keras 2* модел GPT-2 [17].

Табела 2: Поређење перформанси модела GPT-2 и Gemma 3, где М означава милион док В означава милијарду

Карактеристика	GPT-2	Gemma 3
Број параметара	Од 124М до 12,5В	Од 270М до 27В
Контекстни прозор	1024 Токена	8192 Токена
Оптимизација	Није оптимизован за просечан хардвер	Високо оптимизован за просечан хардвер

Након рефакторисања, FL алгоритми, као што је Федеративни просек (FedAvg), могу да прихвате и обрађују *Keras 3* моделе, без потребе за изменама у самој логици алгоритма. Компонента *keras_compat* је ефикасно апстраховала разлике у понашању верзија, осигуравајући да TFF модули увек добију очекиване податке (нпр. сирове тензоре променљивих стања), без обзира на изворну верзију библиотеке *Keras*.

Поред функционалних предности, креирање централизоване компоненте *keras_compat* доноси и значајне предности у погледу одржавања кода. Сва комплексност везана за руковање верзионим разликама је изолована унутар једног модула. Ово чини остатак TFF кода чистијим и олакшава будуће измене, јер свака наредна адаптација захтева измене на само једном, предвидивом месту. Приступ је минимално инвазиван, јер није захтевао дубоке промене у логици постојећих FL алгоритама, већ само замену директних API позива позивима ка *keras_compat* функцијама.

4.3 Дискусија о решењу

Тренутно решења задовољава покриће кода тестовима. Међутим, начин на који су додати тестови за *Keras 3* моделе би могао бити побољшан рефакторисањем тестова за *Keras 2* моделе тако да покривају и случаје коришћена *Keras 3* модела, чиме би се преполовио број тестова. Такође, тест класе би биле прегледније. Мана спајања тест случајева је већа сложеност самих тестова, због различитих понашања библиотеке *Keras*. Из овог разлога изабран је приступ са раздвојеним тест случајевима.

4.4 Утицај рефакторисања TFF екосистема

Током развоја овог решења, радно окружење TFF је ушло у процес великог архитектонског рефакторисања. TFF тим је започео процес одвајања библиотеке на више независних компоненти, што је значајно променило основну структуру кода. Због ове промене, интеграција (*merge*) рефакторисаног кода описаног у овом раду назад у главну грану TFF пројекта тренутно није изводљива без додатних адаптација новој архитектури.

Ипак, ова околност не умањује примарни значај и успешност рада. Рефакторисање и даље у потпуности испуњава своју основну сврху: омогућава учитавање и тренирање напредних Keras 3 модела у окружењу које је било доступно у тренутку израде. Тиме је директно омогућено коришћење модела високих перформанси, као што је Gemma 3, за унапређење предвиђања корисничких претрага у оквиру алата Search Everywhere, што је и била главна мотивација овог рада.

5.1 Резиме рада и остварени циљеви

Примарна мотивација за овај рад проистекла је из потребе за унапређењем функционалности претраге *Search Everywhere* у интегрисаним развојним окружењима, коришћењем модела са побољшаним перформансама. Утврђено је да TFF, радно окружење за федеративно учење које чува приватност података, није подржавало библиотеку *Keras 3*, чиме је онемогућено коришћење модерних модела попут *Gemma 3*.

Главни циљ рада, имплементација паралелне подршке за *Keras 3* уз очување компатибилности са *Keras 2*, је остварен у потпуности. То је постигнуто кроз рефакторисање TFF модула *models*, *optimizers*, *metrics* и *algorithms*. Уведено је архитектонско решење засновано на додавању компоненте *keras_compat*, која апстрахује и изољује сву логику зависну од верзије, док је употреба уније типова у програмском језику *Python* омогућила да TFF функције несметано прихватају објекте из обе верзије библиотеке. Успешност и функционална исправност решења верификована је проласком свих постојећих и новододатих јединичних тестова, чији је број повећан са 140 на 279.

5.2 Кључни доприноси и значај рада

Овај рад доноси два кључна доприноса. Функционални допринос се огледа у томе што је TFF екосистем сада оспособљен за рад са најновијом генерацијом модела. Као што је показано, модели попут *Gemma 3* нуде значајно већи контекстни прозор (*context window*) и оптимизовани су за рад на стандардном хардверу, што је пресудно за апликације у индустрији.

Други, архитектонски допринос, лежи у дизајну минимално инвазивног решења које није нарушило стабилност постојећег TFF кода. Примењени приступ служи као модел за будуће миграције и доказује да је могуће проширити функционалност сложених система отвореног кода без потребе за комплетним рефакторисањем. Тиме је представљен значајан корак ка модернизацији TFF радног окружења.

5.3 Ограничења и будући рад

Током евалуације, препозната су и одређена ограничења. Приступ са раздвојеним тестним случајевима довео је до дуплирања кода за тестирање. Такође, због рефакторисања самог TFF пројекта које се десило паралелно са овим радом, директна интеграција решења у главну грану пројекта тренутно није изводљива.

Наведена ограничења отварају правце за будући рад. Када радно окружење TFF заврши рефакторисање своје архитектуре, биће могуће адаптирати решење из овог рада у новој архитектури, чиме би се омогућила интеграција решења у главну грану пројекта. Овом интеграцијом би корисници радног окружења TFF имали приступ предностима нове верзије библиотеке *Keras* унутар TFF, чиме се отварају врата за истраживање и имплементацију нових, напреднијих алгоритама за федеративно учење која могу искористити флексибилност *multi-backend* архитектуре.

Списак слика

- Слика 1 Пример претраге *Search Everywhere* за термин *compat*. Претрага претражује функционалности IDE, као и пројектну структура како ради проналажења свих појављивања термина претраге унутар пројекта. 1
- Слика 2 Примена FL за предвиђање корисничког уноса на *Gboard Android* тастатури. Мобилни уређај локално тренира над претходним корисничким уносима и ажуриране тежине шаље централном серверу (А). Централни сервер агрегира примљене тежине (В) и шаље свим уређајима побољшан дељени модел (С) [8]. 6

Списак листинга

Листинг 1	Функција <code>_is_compiled</code> из компоненте <code>keras_compat</code>	14
Листинг 2	Функција <code>keras_dtype_to_tf</code> из компоненте <code>keras_compat</code>	15
Листинг 3	Функција <code>is_keras3</code> из компоненте <code>keras_compat</code>	15
Листинг 4	Функција <code>get_optimizer_variables</code> из компоненте <code>keras_compat</code>	16
Листинг 5	Функција <code>ref</code> из компоненте <code>keras_compat</code>	16
Листинг 6	Функција <code>get_variable</code> из компоненте <code>keras_compat</code>	17
Листинг 7	Функција <code>get_variables</code> из компоненте <code>keras_compat</code>	17
Листинг 8	Функција <code>clone_model</code> из компоненте <code>keras_compat</code>	17
Листинг 9	Функција <code>int_shape</code> из компоненте <code>keras_compat</code>	18
Листинг 10	Исечак функције <code>from_keras_model</code> , који приказује имплементацију уније типова.	18
Листинг 11	Исечак функције <code>functional_model_from_keras</code> , који приказује додавање апстракције за добављање променљивих модела преко компоненте <code>keras_compat</code>	19
Листинг 12	Команда за покретање тестова у <i>Bazel</i> инфраструктури, где <code>putanja</code> представља путању до жељене <i>Bazel</i> целине	21

Списак табела

Табела 1	Функције компоненте keras_compat за апстракцију руковања моделом . .	13
Табела 2	Поређење перформанси модела GPT-2 и Gemma 3, где М означава милион док В означава милијарду	22

Списак коришћених скраћеница

Скраћеница	Опис
API	<i>Application Programming Interface</i> (апликациони програмски интерфејс)
B	<i>Billion</i> (милијарда)
CDP	<i>Central Differential Privacy</i> (централна диференцијална приватност)
DP	<i>Differential Privacy</i> (диференцијална приватност)
FedAvg	<i>Federated Averaging</i> (федеративни просек)
FL	<i>Federated Learning</i> (федеративно учење)
GDPR	<i>General Data Protection Regulation</i> (Општа уредба о заштити података)
GPT-2	<i>Generative Pre-trained Transformer 2</i> (модел машинског учења)
IDE	<i>Integrated Development Environment</i> (интегрисано развојно окружење)
LDP	<i>Local Differential Privacy</i> (локална диференцијална приватност)
M	<i>Million</i> (милион)
ML	<i>Machine Learning</i> (машинско учење)
ODP	<i>Optimized Differential Privacy</i> (оптимизована диференцијална приватност)
SE	<i>Search Everywhere</i>
TF	<i>TensorFlow</i>
TFF	<i>TensorFlow Federated</i>

Списак коришћених појмова

Појам	Објашњење
Федеративно учење (FL)	Техника машинског учења која омогућава децентрализовано тренирање модела над подацима који остају на уређајима корисника, чувајући приватност.
Диференцијална приватност (DP)	Математички оквир који гарантује да се присуство или одсуство једног узорка у скупу података не може значајно одразити на излаз анализе, често постигнуто додавањем шума.
Оптимизована диференцијална приватност (ODP)	Варијанта DP прилагођена за машинско учење, која тежи да оптимизује компромис између приватности и тачности модела.
<i>TensorFlow Federated</i> (TFF)	Радно окружење отвореног кода за истраживање и примену федеративног учења и других дистрибуираних израчунавања.
<i>Keras</i>	Библиотека високог нивоа за машинско учење, која служи као интерфејс за позадинска окружења као што су <i>TensorFlow</i> , <i>JAX</i> и <i>PyTorch</i> .
Модел (ML)	Математичка репрезентација система или процеса, научена из података, која се користи за предвиђање или класификацију.
Оптимизатор (ML)	Алгоритам који се користи током тренирања модела за ажурирање његових параметара (тежина) како би се минимизирала функција грешке.
Метрика (ML)	Мера која се користи за евалуацију перформанси модела (нпр. тачност, губитак).
Агрегација (у FL)	Процес на централном серверу где се ажурирања модела (тежине) прикупљена од клијената комбинују како би се направио побољшани глобални модел.
Клијент (у FL)	Уређај (нпр. мобилни телефон, рачунар) који учествује у федеративном учењу, тренирајући локални модел на својим подацима.
Сервер (у FL)	Централна компонента у федеративном учењу која координира процес, агрегира ажурирања од клијената и дистрибуира ажурирани глобални модел.
Стање сервера (у FL)	Скуп параметара који дефинишу тренутно стање FL процеса на серверу, укључујући тежине глобалног модела и стање оптимизатора.
Итеративни процес (у TFF)	Основна структура у TFF-у која дефинише један корак (рунду) федеративног алгоритма, укључујући иницијализацију и ажурирање стања.

<i>Stateless</i> парадигма	Пристап у програмирању где компонента не чува интерно стање између позива (нпр. <i>Keras</i> 3 оптимизатори). Стање се експлицитно прослеђује као аргумент.
<i>Stateful</i> парадигма	Пристап где компонента интерно чува и управља својим стањем између позива (нпр. <i>Keras</i> 2 оптимизатори).
Унија типова (<i>Python</i>)	Могућност у програмском језику <i>Python</i> да се дефинише да променљива може припадати једном од више наведених типова.

Биографија

Марко Митошевић је рођен у Суботици 2002. године. Након завршене гимназије “Исидора Секулић” у Новом Саду, уписује смер Софтверско инжењерство и информационе технологије на Факултету техничких наука у Новом Саду. Стручну праксу је обавио у компанији *JetBrains*, где се након праксе и запослио. Ради на позицији софтверског инжењера у департману за истраживања. Током студија је волонтирао у студенским организацијама Ерасмус студентска мрежа (*Erasmus Student Network, ESN*), Европска асоцијација студената електро инжењерства (*Electrical Engineering Students' European assoCiation, EESTEC*), као и у Европском студентском форуму (*Association des États Généraux des Étudiants de l'Europe, AEGEE*), служећи један мандат као председник огранка у Новом Саду. Такође је активан члан невладине организације *Capre Noctrem*, еколошког удружења које за главни фокус има проблематику светлосног загађења, заштиту природно тамних подручја и разбој астрономског туризма. Био је амбасадор за младе у склопу програма Омладинска престоница Европе Нови Сад (*OPENS*).

Литература

- [1] JetBrains, „IntelliJ IDEA“. Приступљено: 12. Октобар 2025. [На Интернету]. Доступно на <https://www.jetbrains.com/idea>
- [2] E. Union, „Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance)“. Приступљено: 12. Октобар 2025. [На Интернету]. Доступно на <https://eur-lex.europa.eu/eli/reg/2016/679/oj/eng>
- [3] G. Cloud, „Federated learning: a guide to what it is and how it works“. Приступљено: 12. Октобар 2025. [На Интернету]. Доступно на <https://cloud.google.com/discover/what-is-federated-learning?hl=en>
- [4] M. A. I. U. D. T. Q. Maria Iqbal Asadullah Tariq, „FL-ODP: An Optimized Differential Privacy Enabled Privacy Preserving Federated Learning“. Приступљено: 12. Октобар 2025. [На Интернету]. Доступно на <https://ieeexplore.ieee.org/document/10287349>
- [5] Google, „TensorFlow Federated: Machine Learning on Decentralized Data“. Приступљено: 12. Октобар 2025. [На Интернету]. Доступно на <https://www.tensorflow.org/federated>
- [6] Google, „Gemma 3“. Приступљено: 24. Октобар 2025. [На Интернету]. Доступно на <https://deepmind.google/models/gemma/gemma-3>
- [7] Meta, „Open-source AI Models for Any Application | Llama 3“. Приступљено: 21. Октобар 2025. [На Интернету]. Доступно на <https://www.llama.com/models/llama-3>
- [8] R. M. S. R. F. B. S. A. H. E. C. K. D. R. Andrew Hard Kanishka Rao, „Federated Learning for Mobile Keyboard Prediction“. Приступљено: 12. Октобар 2025. [На Интернету]. Доступно на <https://arxiv.org/abs/1811.03604>
- [9] C. Dwork, „Differential Privacy“. Приступљено: 16. Октобар 2025. [На Интернету]. Доступно на https://doi.org/10.1007/11787006_1
- [10] A. R. Cynthia Dwork, *The Algorithmic Foundations of Differential Privacy*. University of Pennsylvania, 2014.

-
- [11] I. G. B. M. I. M. K. T. L. Z. Martín Abadi Andy Chu, „Deep Learning with Differential Privacy“. Приступљено: 19. Октобар 2025. [На Интернету]. Доступно на https://link.springer.com/chapter/10.1007/978-3-540-79228-4_1
- [12] E. D. C. U. C. L. D. A. T. I. Mohammad Naseri Jamie Hayes, „Local and Central Differential Privacy for Robustness and Privacy in Federated Learning“. Приступљено: 19. Октобар 2025. [На Интернету]. Доступно на <https://arxiv.org/abs/2009.03561>
- [13] Keras, „Keras: Deep Learning for humans“. Приступљено: 21. Октобар 2025. [На Интернету]. Доступно на <https://keras.io/>
- [14] Kaggle, „Kaggle: Your Machine Learning and Data Science Community“. Приступљено: 21. Октобар 2025. [На Интернету]. Доступно на <https://www.kaggle.com/>
- [15] Python, „The official home of the Python Programming Language“. Приступљено: 22. Октобар 2025. [На Интернету]. Доступно на <https://www.python.org/>
- [16] Bazel, „Bazel“. Приступљено: 24. Октобар 2025. [На Интернету]. Доступно на <https://bazel.build/>
- [17] H. Face, „GPT-2“. Приступљено: 24. Октобар 2025. [На Интернету]. Доступно на <https://huggingface.co/openai-community/gpt2>