



# УНИВЕРЗИТЕТ У НОВОМ САДУ ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА



УНИВЕРЗИТЕТ У НОВОМ САДУ

ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

НОВИ САД

Депарتمان за рачунарство и аутоматику

Одсек за рачунарску технику и рачунарске комуникације

## ЗАВРШНИ (BACHELOR) РАД

Кандидат: Марко Николовски

Број индекса: РА 140/2018

Тема рада: Дистрибуиране апликације за обучавање неуралне мреже *MNIST*  
на окружењима *PTB-FLA* и *MPT-FLA*

Ментор рада: Проф. др Мирослав Поповић

Нови Сад, Јул, 2024.



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

## КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, <b>РБР</b> :	
Идентификациони број, <b>ИБР</b> :	
Тип документације, <b>ТД</b> :	Монографска документација
Тип записа, <b>ТЗ</b> :	Текстуални штампани материјал
Врста рада, <b>ВР</b> :	Завршни (Bachelor) рад
Аутор, <b>АУ</b> :	<b>Марко Николовски</b>
Ментор, <b>МН</b> :	<b>Проф. др Мирослав Поповић</b>
Наслов рада, <b>НР</b> :	<b>Дистрибуиране апликације за обучавање неуралне мреже MNIST на окружењима PTB-FLA и MPT-FLA</b>
Језик публикације, <b>ЈП</b> :	Српски / ћирилица
Језик извода, <b>ЈИ</b> :	Српски
Земља публикавања, <b>ЗП</b> :	Република Србија
Уже географско подручје, <b>УГП</b> :	Војводина
Година, <b>ГО</b> :	<b>2024.</b>
Издавач, <b>ИЗ</b> :	Ауторски репринт
Место и адреса, <b>МА</b> :	Нови Сад; трг Доситеја Обрадовића 6
Физички опис рада, <b>ФО</b> : (поглавља/страна/ цитата/табела/слика/графика/прилога)	<b>7/31/0/2/11/0/0</b>
Научна област, <b>НО</b> :	Електротехника и рачунарство
Научна дисциплина, <b>НД</b> :	Рачунарска техника
Предметна одредница/Кључне речи, <b>ПО</b> :	<b>IoT, Неуралне мреже, Федеративно учење, PTB-FLA, MPT-FLA</b>
<b>УДК</b>	
Чува се, <b>ЧУ</b> :	У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, <b>ВН</b> :	
Извод, <b>ИЗ</b> :	У савременом свету вештачке интелигенције неуралне мреже играју кључну улогу у обради података. Овај рад приказује развој две апликације за обучавање неуралне мреже коришћењем MNIST скупа података на два различита окружења. Прва апликација је развијена и тестирана на PTB-FLA окружењу, а друга апликација је развијена адаптирањем и тестирањем прве апликације за MicroPython-у и RPi Pico W плочицу.
Датум прихватања теме, <b>ДП</b> :	
Датум одбране, <b>ДО</b> :	
Чланови комисије, <b>КО</b> :	Председник: проф. др. Иван Каштелан
	Члан: доц. др Миодраг Ђукић
	Члан, ментор: проф. др Мирослав Поповић
	Потпис ментора




UNIVERSITY OF NOVI SAD • FACULTY OF TECHNICAL SCIENCES  
21000 NOVI SAD, Trg Dositeja Obradovića 6

## KEY WORDS DOCUMENTATION

Accession number, <b>ANO</b> :			
Identification number, <b>INO</b> :			
Document type, <b>DT</b> :	Monographic publication		
Type of record, <b>TR</b> :	Textual printed material		
Contents code, <b>CC</b> :	Bachelor Thesis		
Author, <b>AU</b> :	Marko Nikolovski		
Mentor, <b>MN</b> :	Miroslav Popović, PhD		
Title, <b>TI</b> :	Distributed applications for training <i>MNIST</i> neural network on <i>PTB-FLA</i> and <i>MPT-FLA</i> federated learning frameworks		
Language of text, <b>LT</b> :	Serbian		
Language of abstract, <b>LA</b> :	Serbian		
Country of publication, <b>CP</b> :	Republic of Serbia		
Locality of publication, <b>LP</b> :	Vojvodina		
Publication year, <b>PY</b> :	2024.		
Publisher, <b>PB</b> :	Author's reprint		
Publication place, <b>PP</b> :	Novi Sad, Dositeja Obradovica sq. 6		
Physical description, <b>PD</b> : (chapters/pages/ref./tables/pictures/graphs/appendixes)	7/31/0/2/11/0/0		
Scientific field, <b>SF</b> :	Electrical Engineering		
Scientific discipline, <b>SD</b> :	Computer Engineering, Engineering of Computer Based Systems		
Subject/Key words, <b>S/KW</b> :	<i>IoT, Neural network, Federated learning, PTB-FLA, MPT-FLA</i>		
<b>UC</b>			
Holding data, <b>HD</b> :	The Library of Faculty of Technical Sciences, Novi Sad, Serbia		
Note, <b>N</b> :			
Abstract, <b>AB</b> :	In the modern world of artificial intelligence, neural networks play a crucial role in data processing. This paper presents the development of two applications for training a neural network using the <i>MNIST</i> dataset on two different frameworks. The first application was developed and tested on a <i>PTB-FLA</i> framework and second application was developed by adapting and testing of the first application for the <i>MicroPython</i> and <i>RPi Pico W</i> .		
Accepted by the Scientific Board on, <b>ASB</b> :			
Defended on, <b>DE</b> :			
Defended Board, <b>DB</b> :	President:	Ivan Kaštelan, PhD	Menthor's sign
	Member:	Miodrag Đukić, PhD	
	Member, Mentor:	Miroslav Popović, PhD	



	УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Број:
	<b>ЗАДАТАК ЗА ЗАВРШНИ РАД</b>	Датум:

Студијски програм:	Рачунарство и аутоматика		
Студент:	Марко Николовски	Број индекса:	РА 140/2018
Степен и врста студија:	први степен, основне академске студије		
Област:	Електротехника и рачунарство		
Ментор:	проф. др Мирослав Поповић		
<p>НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ЗАВРШНИ РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:</p> <ul style="list-style-type: none"> <li>- проблем – тема рада;</li> <li>- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;</li> </ul>			

### Наслов завршног рада:

**Дистрибуиране апликације за обучавање неуралне мреже *MNIST* на окружењима *PTB-FLA* и *MPT-FLA***

### Текст задатка

У оквиру овог дипломског рада потребно је урадити следеће:

1. Упознати се са окружењима *PTB-FLA* и *MPT-FLA*.
2. Упознати се са секвенцијалним програмом за обучавање *MNIST* неуралне мреже.
3. Користећи *PTB-FLA* развојну парадигму са четири фазе, развити дистрибуирану апликацију за обучавање *MNIST* неуралне мреже на окружењу *PTB-FLA*.
4. Тестирати и еволуирати *PTB-FLA* апликацију на *PC* рачунару.
5. Развити дистрибуирану апликацију за обучавање *MNIST* неуралне мреже на окружењу *MPT-FLA* адаптирањем *PTB-FLA* апликације на *MicroPython* и *RPi Pico W* плочицу.
6. Тестирати и еволуирати *MPT-FLA* апликацију на мрежи са *PC* рачунарима и *RPi Pico W* плочицама.
7. Написати текст дипломског рада.

Руководилац студијског програма:	Ментор рада:

Примерак за: ☐ - Студента; ☐ - Ментора

## **Захвалност**

Најискреније захваљујем професору Мирославу Поповићу на неизмерној подршци и стручном вођству током израде мог дипломског рада.

Такође, изражавам дубоку захвалност својим најближима на њиховој подршци и разумевању током целих студија.

## САДРЖАЈ

<b>1. Увод .....</b>	<b>10</b>
<b>2. Теоријске основе .....</b>	<b>11</b>
2.1 Федеративно учење .....	11
2.1.1 Типови федеративног учења .....	11
2.2 Неуралне мреже .....	12
2.2.1 Модел вештачког неурона .....	12
2.2.2 Архитектура неуралне мреже .....	13
2.2.3 Обучавање неуралне мреже .....	13
2.3 Увод у <i>PTB-FLA</i> .....	14
2.3.1 Karakteristike <i>PTB-FLA</i> .....	14
2.3.2 Архитектура <i>PTB-FLA</i> система .....	14
2.3.3 <i>PTB-FLA API</i> .....	15
2.4 Увод у <i>MPT-FLA</i> .....	15
2.4.1 Карактеристике <i>MPT-FLA</i> .....	15
2.4.2 Архитектура <i>MPT-FLA</i> система .....	16
2.4.3 <i>MPT-FLA API</i> .....	16
2.5 Технологије .....	17
2.5.1 <i>Python</i> .....	17
2.5.2 <i>Micropython</i> .....	17
<b>3. Концепт решења .....</b>	<b>18</b>
3.1 Кораки у реализацији апликација .....	18
<b>4. Програмско решење .....</b>	<b>19</b>
4.1 Секвенцијални код за обучавање <i>MNIST</i> неуралне мреже .....	19
4.1.1 <i>MNIST</i> скуп података .....	19
4.1.2 Припрема улазног скупа података за обучавање и тестирање .....	20

4.1.3	Архитектура <i>MNIST</i> неуралне мреже.....	20
4.1.4	Модел <i>MNIST</i> неуралне мреже.....	21
4.1.5	Обучавање <i>MNIST</i> неуралне мреже.....	21
4.1.6	Тестирање <i>MNIST</i> неуралне мреже.....	22
4.2	Развој дистрибуиране апликације на окружењу <i>PTB-FLA</i> .....	23
4.2.1	Подешавање окружења за рад.....	23
4.2.2	Четири фазе <i>PTB-FLA</i> развојне парадигме .....	23
4.2.3	Покретање дистрибуиране апликације на <i>PTB-FLA</i> окружењу.....	25
4.3	Развој дистрибуиране апликације на окружењу <i>MTB-FLA</i> .....	25
4.3.1	Подешавање окружења за рад.....	25
4.3.2	Прилагођавање <i>PTB-FLA</i> апликације за <i>MicroPython</i> .....	25
4.3.3	Покретање <i>MTB-FLA</i> апликације.....	26
<b>5.</b>	<b>Испитивање.....</b>	<b>27</b>
5.1	Резултати тестирања.....	27
5.1.1	Зависност прецизности коначног модела од величине улазног скупа података сервера.....	27
5.1.2	Зависност прецизности коначног модела од броја итерација сервера.....	28
<b>6.</b>	<b>Закључак .....</b>	<b>30</b>
<b>7.</b>	<b>Литература .....</b>	<b>31</b>



## СПИСАК СЛИКА

*Слика 2.1 Вештачки неурон*

*Слика 2.2 Вишеслојна неурална мрежа са простирањем сигнала унапред*

*Слика 4.1 Део кода који описује поступак припреме улазног скупа података*

*Слика 4.2 Функција која дефинише величине слојева неуралне мреже*

*Слика 4.3 Функција која дефинише параметре почетног модела неуралне мреже*

*Слика 4.4 Функција која описује поступак обучавања MNIST неуралне мреже*

*Слика 4.5 Функција која описује поступак тестирања MNIST неуралне мреже*

*Слика 4.6 Клијентска и серверска функција повратног позива*

*Слика 4.7 Пример конфигурационе датотеке*

*Слика 5.1 График зависности прецизности коначног модела од величине улазног скупа података сервера*

*Слика 5.2 График зависности прецизности коначног модела од броја итерација сервера*

## СПИСАК ТАБЕЛА

*Табела 5.1 Зависност прецизности коначног модела од величине улазног скупа података сервера*

*Табела 5.2 Зависност прецизности коначног модела од броја итерација сервера*

## СКРАЋЕНИЦЕ

**MNIST** - Улазни скуп података за препознавање руком писаних цифара

**PTB-FLA** - *Python* окружење за развој алгоритама федеративног учења

**MPT-FLA** - *MicroPython* окружење за развој алгоритама федеративног учења

**IoT** - *Internet of Things*, интернет ствари

**I/O** - *Input/Output*, улаз/излаз

**API** - Програмска спега апликације

**CMD** - *Command Prompt*, командна линија

**IDE** - Интегрисано развојно окружење

## 1. Увод

У савременом свету вештачке интелигенције, неуралне мреже играју кључну улогу у обради података, а *MNIST* скуп података са ручно писаним цифрама је стандард за тестирање ових алгоритама. Циљ овог дипломског рада је развој апликације за обучавање неуралне мреже коришћењем *MNIST* скупа, уз примену два различита окружења.

Прво, апликација ће бити развијена и тестирана у *PTB-FLA* (eng. *Python Testbed for Federated Learning Algorithms*) систему. *PTB-FLA* је једноставно окружење у чистом *Python-у*, које омогућава развој и тестирање алгоритама федеративног учења, како централизованих тако и децентрализованих. Затим ће апликација бити прилагођена за рад на *MicroPython-у* и уређајима са ограниченим ресурсима, у нашем случају *RPi Pico W*, користећи окружење *MPT-FLA* (eng. *MicroPython Testbed for Federated Learning Algorithms*). Овај приступ ће истражити изазове и предности примене федеративног учења на *IoT* уређајима и системима на ивици. На крају рада биће анализирани резултати и поређење перформанси обучавања неуронске мреже.

Овај рад ће показати како се алгоритми машинског учења могу ефикасно примењивати на различитим платформама и уређајима, отварајући нове могућности за примену у *IoT* и системима на ивици.

У другом поглављу истражићемо теоријске основе које су кључне за разумевање теме овог дипломског рада. Треће поглавље посвећено је концепту решења, где ћемо детаљно описати кораке потребне за имплементацију апликације. Четврто поглавље ће се фокусирати на софтверско решење, где ћемо приказати конкретан код за обуку *MNIST* неуралне мреже, као и процес развоја дистрибуиране апликације на окружењима *PTB-FLA* и *MPT-FLA*. У петом поглављу, испитаћемо имплементирано решење и анализирати добијене резултате.

## 2. Теоријске основе

У овом поглављу ћемо изложити теоријске основе које су кључне за разумевање федеративног учења и неуронских мрежа. Такође ћемо се упознати са окружењима за федеративно учење *PTB-FLA* и *MPT-FLA*. Осим тога, истражићемо и технологије које користимо за развој софтвера за обучавање неуралне мреже, како бисмо боље разумели њихову улогу и примену у овом дипломском раду.

### 2.1 Федеративно учење

Федеративно учење је напредна методологија машинског учења која омогућава обучавање, тестирање и употребу (тзв. закључивање) модела на дистрибуираним уређајима или серверима без потребе за централизовањем података. Модел машинског учења се тренира на различитим уређајима који поседују локалне податке, без потребе да се ти подаци шаљу на централни сервер. Уместо тога, локални уређаји шаљу само ажурирања модела, као што су тежине или градијенти, назад на централни сервер. Централни сервер тада агрегира ова ажурирања како би унапредио глобални модел. Ова техника омогућава очување приватности података, побољшање ефикасности и прилагођавање модела специфичним корисницима или уређајима.

#### 2.1.1 Типови федеративног учења

- **Централизовано федеративно учење** укључује централни сервер који оркестрира процес обуке. Учесници (клијенти) обучавају локалне моделе на својим подацима и шаљу ажуриране параметре модела централном серверу. Сервер агрегира ове параметре како би ажурирао глобални модел, који се затим дистрибуира назад клијентима.
- **Децентрализовано федеративно учење** елиминише потребу за централним сервером. Уместо тога, учесници директно комуницирају једни са другима како би делили и агрегирали ажурирања модела. Ова комуникација између равноправних чворова (енг. *peer-to-peer*) осигурава да не постоји једна тачка отказа и побољшава приватност дистрибуцијом процеса агрегирања.

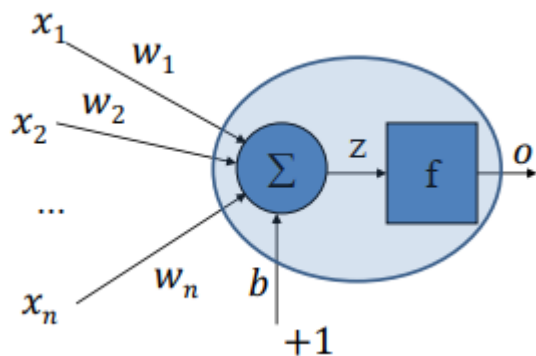
- **Хоризонтално федеративно учење** се дешава када скупови података код различитих учесника деле исти простор карактеристика, али се разликују у узорцима. У суштини, скупови података имају исту структуру (карактеристике), али садрже податке о различитим ентитетима.
- **Вертикално федеративно учење** се дешава када учесници имају скупове података који деле исти простор узорака, али се разликују у карактеристикама. То значи да сваки учесник има различите атрибуте о истом скупу ентитета.

## 2.2 Неуралне мреже

Неуралне мреже представљају основну компоненту дубоког учења. Ове мреже су инспирисане биолошким неуронима и састоје се од вештачких неурона повезаних у слојевима. Сваки неурон прима улазне податке, обрађује их и прослеђује резултат следећем слоју неурона. Неуралне мреже су изузетно способне за препознавање образаца и доношење одлука на основу података.

### 2.2.1 Модел вештачког неурона

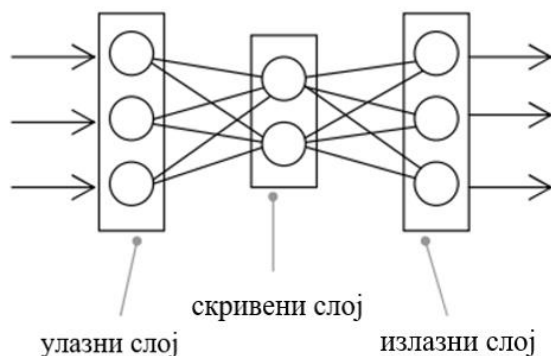
Основна компонента неуралних мрежа је неурон. Неурон је јединица која обрађује информације примљене на улазу и даје један излаз. Састоји се од улазних сигнала  $X_i$ , синаптичких тежина  $W_i$ , прага  $b$ , активационих функција  $f$ , сабирача и једног излаза  $o$  (као што је приказано на слици 2.1). Активационе функције које се користе у неуронима могу бити различите, укључујући линеарну функцију, праговску функцију, полу-линеарну функцију, хиперболички тангенс,  $ReLU$  и  $ELU$ . Излаз  $o$  се рачуна помоћу формуле:  $o = f(z) = f(\sum W_i X_i + b)$



Слика 2.1 Вештачки неурон

### 2.2.2 Архитектура неуралне мреже

Неурони у мрежи груписани су у слојевима. У зависности од улоге у мрежи, разликујемо улазни слој који прима податке из околине, излазни слој који даје резултате обраде и скривени слој који се налази између улазног и излазног слоја. Везе у мрежи су успостављене између неурона из различитих слојева. Веза између слојева је једносмерна веза унапред (енг. *feedforward*). На слици 2.2 је приказана архитектура неуралне мреже са простирањем сигнала унапред.



Слика 2.2 Вишеслојна неурална мрежа са простирањем сигнала унапред

### 2.2.3 Обучавање неуралне мреже

Процес обучавања неуралних мрежа почиње пропагацијом унапред, где се улазни подаци пропуштају кроз мрежу. Улази се множе са тежинама и пролазе кроз активационе функције, што резултира излазним вредностима. Ове излазне вредности се затим пореде са стварним вредностима помоћу функције циља  $J(W,b)$ . Функција циља  $J$  типично представља разлику жељеног и оствареног излаза мреже. Циљ обучавања је минимизирање вредности функције циља (губитка). Пропагација уназад је кључни алгоритам за обучавање неуралних мрежа. Након што се израчуна губитак, овај алгоритам рачуна градијенте губитка у односу на тежине. Ови градијенти се затим користе за ажурирање тежина мреже. Градијентни спуст је оптимизациони алгоритам који се користи за минимизирање функције циља. Обучавање се спроводи итеративно у више циклуса (епоха). Током сваке епохе, мрежа пролази кроз целокупан скуп података за обучавање, постепено побољшавајући своје перформансе.

## 2.3 Увод у *PTB-FLA*

Постојећи оквири за федеративно учење често се суочавају са изазовима приликом примене на *IoT* уређајима и системима на ивици, због сложене инсталације и зависности од специфичних оперативних система и инфраструктуре облака.

У раду [1] представљен је *PTB-FLA*, једноставно окружење развијено у *Python*-у. Ово окружење је пројектовано за развој и тестирање алгоритама федеративног учења, са посебним фокусом на примену на *IoT* уређајима и системима на ивици.

### 2.3.1 Karakteristike *PTB-FLA*

*PTB-FLA* је написан у чистом *Python*-у, што значи да зависи само од стандардних *Python* пакета. Овакав приступ осигурава минимални отисак апликације, што је одговарајуће за *IoT* уређаје. Са минималним зависностима, инсталација апликације је поједностављена. *PTB-FLA* подржава и централизоване и децентрализоване алгоритме федеративног учења, што омогућава флексибилно коришћење различитих топологија.

### 2.3.2 Архитектура *PTB-FLA* система

Архитектура *PTB-FLA* система састоји се од три главна слоја: дистрибуирани апликациони слој, *PTB-FLA* слој и *Python* слој. Дистрибуирани апликациони слој обухвата апликационе модуле и конзолну скрипту за покретање. Модул *launcher* покреће дистрибуирану апликацију састављену од независних процеса, где се сваки процес извршава у засебном терминалу. *PTB-FLA* слој садржи класу *PtbFla* и модуле *trapi* и *launcher*. Апликациони модул користи *PtbFla API* за формирање и уништавање *testbed* инстанци и за извршавање дистрибуираних алгоритама. *API* функције *fl\_centralized* и *fl\_decentralized* користе модул *trapi* за комуникацију са другим инстанцама. *Python* слој укључује класе из пакета *multiprocessing* и *subprocess*.



### 2.3.3 PTB-FLA API

API PTB-FLA система састоји се од неколико кључних функција које омогућавају рад са федеративним учењем у централизованом и децентрализованом окружењу:

- Функција **PtbFla(noNodes, nodeId, flSrvId=0)** је конструктор који се позива као глобална функција и нема повратну вредност.
- Функција **fl\_centralized(sfun, cfun, ldata, pdata, noIters=1)** служи за извођење алгорита централизованог федеративног учења. Повратна вредност је крајња верзија локалних података чвора.
- Функција **fl\_decentralized(sfun, cfun, ldata, pdata, noIters=1)** намењена је за извођење алгорита децентрализованог федеративног учења. Повратна вредност је, као и код *fl\_centralized*, крајња верзија локалних података чвора.
- Функција **PtbFla()** је деструктор који се имплицитно позива од стране *garbage collector*а или експлицитно приликом брисања објекта. Ова функција нема параметре и не враћа никакву вредност.

## 2.4 Увод у MPT-FLA

У раду [2] представљено је ново окружење за алгоритме федеративног учења. *MPT-FLA* превазилази ограничења свог претходника *PTB-FLA*. Док је *PTB-FLA* био ограничен за покретање инстанци апликација на једном рачунару, *MPT-FLA* омогућава овим инстанцама да раде на различитим мрежним чворовима, као што су рачунари и *IoT* уређаји, чинећи га посебно погодним за системе на ивици.

### 2.4.1 Карактеристике MPT-FLA

*MPT-FLA* доноси значајна унапређења у односу на *PTB-FLA*. Оквир је развијен са циљем да омогући рад апликација на различитим мрежним чворовима, укључујући *IoT* уређаје. Кључна карактеристика је употреба асинхроних *I/O* апстракција које омогућавају замену традиционалних процеса са корутинама, чиме се постиже ефикаснија комуникација и обрада података у реалном времену. Ова архитектура омогућава покретање *MPT-FLA* апликација на уређајима са ограниченим ресурсима, као што су микроконтролери, уз минималне измене у коду.

### 2.4.2 Архитектура *MPT-FLA* система

Архитектура *MPT-FLA* је изграђена на *asyncio* апстракцијама *Python-a*, што омогућава ефикасно управљање корутинским задацима и догађајима. Први корак у развоју *MPT-FLA* укључивао је замену локалних серверских процеса са нитима и редова са листама. Други корак је подразумевао интеграцију *asyncio* апстракција, замену нити са корутинама и редова са *asyncio* догађајима. Ова архитектура омогућава робустан и ефикасан рад на различитим платформама, укључујући и оне са ограниченим хардверским ресурсима.

### 2.4.3 *MPT-FLA API*

*API MPT-FLA* система састоји се од неколико кључних функција које омогућавају рад са федеративним учењем у централизованом и децентрализованом окружењу.

- Функција **PtbFla(noNodes, nodeId, flSrvId=0, mrIpAdr='localhost')** је конструктор који се користи за иницијализацију чвора у *MPT-FLA* систему и нема повратну вредност.
- Функција **async start()** покреће чвор и иницира његову улогу у покретању целог система. Ова функција такође нема повратну вредност.
- Функција **async fl\_centralized(sfun, cfun, ldata, pdata, noIters=1)** извршава алгоритам централизованог федеративног учења. Повратна вредност је коначна верзија локалних података чвора.
- Функција **async fl\_decentralized(sfun, cfun, ldata, pdata, noIters=1)** извршава алгоритам децентрализованог федеративног учења. Повратна вредност је такође коначна верзија локалних података чвора.
- Функција **async get1Meas(peerId, odata)** омогућава размену података са паром (*peer*) у текућем *TDM (Time Division Multiplexing)* временском слоту. Повратна вредност су примљени подаци од пара (*peer*).

## 2.5 Технологије

У свету савременог софтверског развоја, програмски језици играју кључну улогу у омогућавању креирања разноврсних апликација и система. Два истакнута представника у овом домену које смо користили су *Python* и његова минимална имплементација *MicroPython*. Оба језика имају широку примену и нуде разноврсне могућности, али су прилагођени различитим типовима задатака и окружењима. У овом одељку, детаљно ћемо истражити обе технологије, њихове карактеристике и примене.

### 2.5.1 *Python*

*Python* је програмски језик високог нивоа који се извршава интерпретирањем и одликује се једноставношћу и читљивошћу синтаксе. Данас је један од најпопуларнијих језика за развој софтвера, веб апликација, научног рачунарства, вештачке интелигенције и аутоматизације. *Python* подржава објектно оријентисано програмирање, што омогућава структурирање кода кроз објекте и класе. Променљиве у *Python-у* немају унапред дефинисане типове података, већ се тип одређује динамички током извршавања програма. *Python* има активну заједницу која континуирано развија библиотеке, модуле и алате који олакшавају развој апликација. Примена *Python-а* је веома широка и обухвата многе области. За научно рачунарство, библиотеке као што су *NumPy*, *Pandas* и *Matplotlib* омогућавају анализу података, обраду слика и визуализацију резултата. У области вештачке интелигенције, *TensorFlow*, *PyTorch* и *Keras* су популарне библиотеке за развој модела дубоког учења и вештачке интелигенције.

### 2.5.2 *Micropython*

*MicroPython* је минимална имплементација *Python-а* прилагођена за микроконтролере и уграђене системе. Користи већину синтаксних могућности стандардног *Python-а*, што олакшава програмерима који су већ упознати са *Python-ом* да лако пређу на рад са овим језиком. Оптимизован је за рад на уређајима као што су микроконтролери (ми смо користили *RPi Pico W*), што омогућава ефикасно коришћење ограничених ресурса тих уређаја. Примена *MicroPython-а* је веома широка и обухвата многе области. У области *Internet ствари* (eng. *Internet of Things*), *MicroPython* се често користи за развој *IoT* уређаја због једноставности и могућности брзе израде прототипова. У образовању је идеалан за учење програмирања и упознавање са електроником, захваљујући својој једноставности и директној примени на хардверу.

## 3. Концепт решења

У овом поглављу описујемо концептуални план за имплементацију дистрибуираних апликација за обуку неуралне мреже користећи *PTB-FLA* и *MPT-FLA* окружења.

### 3.1 Кораци у реализацији апликација

Први корак у реализацији апликација је креирање секвенцијалног кода за обуку *MNIST* неуралне мреже. Ово укључује дефинисање архитектуре неуралне мреже, припрему улазног скупа података, имплементацију процеса обуке и тестирања модела. Секвенцијални код служи као основа за даљу дистрибуцију и прилагођавање у *PTB-FLA* и *MPT-FLA* окружењима.

Користећи *PTB-FLA* развојну парадигму са четири фазе, следећи корак је развијање дистрибуиране апликације за обуку *MNIST* неуралне мреже на *PTB-FLA* платформи. Ово укључује адаптацију секвенцијалног кода за рад у дистрибуираном окружењу, поделу улазног скупа података на засебне делове за сваког клијента, имплементацију функција повратног позива (*eng. callback*) које омогућавају комуникацију између клијената и сервера, као и осигурање правилног агрегирања резултата.

Након развоја дистрибуиране апликације на *PTB-FLA* платформи, следи тестирање апликације на локалном рачунару (*localhost*). Овај корак обухвата верификацију функционалности апликације, перформансе у реалним условима, као и оптимизацију кода према потребама дистрибуираног окружења. Циљ је осигурати стабилност и ефикасност апликације пре преласка на следећу фазу.

Након успешног тестирања *PTB-FLA* апликације, следећи корак је адаптација и развој дистрибуиране апликације за обуку *MNIST* неуралне мреже у *MPT-FLA* окружењу. Ово подразумева прилагођавање апликације за рад у *MicroPython* окружењу, уз коришћење *RPi Pico W* плочица. Главни изазов у овој фази је оптимизација кода за рад на микроконтролерима са ограниченим ресурсима.

Последњи корак у реализацији апликације је тестирање *MPT-FLA* апликације на мрежи која укључује *PC* рачунаре и *RPi Pico W* плочице. Овај корак обухвата проверу функционалности и перформанси у стварном мрежном окружењу, идентификацију и решавање потенцијалних проблема, као и даљу оптимизацију апликације за ефикасан рад у дистрибуираном окружењу.

## 4. Програмско решење

Програмско решење представља главни део овог дипломског рада, где се практично примењују теоријска знања из области федеративног учења и неуралних мрежа. У овом поглављу, детаљно ћемо описати имплементацију различитих компоненти система, укључујући секвенцијални код за обуку *MNIST* неуралне мреже, као и развој дистрибуираних апликација у окружењима *PTB-FLA* и *MPT-FLA*.

### 4.1 Секвенцијални код за обучавање *MNIST* неуралне мреже

У овом делу ћемо се фокусирати на имплементацију секвенцијалног кода за обуку неуралне мреже на *MNIST* улазном скупу података. Детаљно ћемо описати процес изградње, обуке и тестирања *MNIST* неуралне мреже, укључујући дефинисање модела, припрему улазних података и тестирање перформанси.

#### 4.1.1 *MNIST* скуп података

*MNIST* (*Modified National Institute of Standards and Technology*) је широко коришћен улазни скуп података за препознавање ручно писаних цифара. Стандардна верзија садржи слике величине 28x28 пиксела, где свака слика представља цифру од 0 до 9. Слике су у сивим тоновима са вредностима пиксела у распону од 0 до 255. Због своје једноставности и доступности, *MNIST* је постао стандард за тестирање алгоритама за препознавање ручно писаних цифара.

### 4.1.2 Припрема улазног скупа података за обучавање и тестирање

Припрема улазног скупа података је почетни корак у процесима обуке и тестирања неуралне мреже. Овај поступак укључује поделу скупа података на два дела: један за обуку ( $X_{train}$ ) и други за тестирање ( $X_{test}$ ). Ова подела омогућава процену перформанси модела на подацима који нису коришћени током обуке. За сваки скуп података чувамо одговарајуће ознаке ( $Y_{train}$  и  $Y_{test}$ ) како бисмо могли правилно проценити тачност модела. Улазни подаци се нормализују дељењем вредности пиксела са 255, чиме се пиксели доводе у опсег од 0 до 1. У нашем случају, улазни скуп података садржи 5000 узорака. Првих 1000 узорака користимо за тестирање, док преостале узорке користимо за обуку. На слици 4.1 је приказан део кода који описује овај поступак.

```
m, n = data.shape

test_data = data[0:1000].T
Y_test = test_data[0]
X_test = test_data[1:n]
X_test = X_test / 255

train_data = data[1000:m]
train_data = train_data.T

Y_train = train_data[0]
X_train = train_data[1:n]
X_train = X_train / 255

Y_train = Y_train.T
Y_test = Y_test.T

Y_train = np.array([Y_train])
Y_test = np.array([Y_test])
```

Слика 4.1 Део кода који описује поступак *postupak pripreme* улазног скупа података

### 4.1.3 Архитектура **MNIST** неуралне мреже

Архитектура неуралне мреже се прилагођава улазном скупу података који користимо. За стандардни **MNIST** скуп података, улазни слој неуралне мреже има 784 неурона, што одговара величини слика од 28x28 пиксела. Излазни слој има 10 неурона, што одговара броју цифара које класификујемо. За скривени слој можемо одабрати различит број неурона, у нашем случају користили смо 128 неурона. На слици 4.2 је приказана функција која дефинише величине слојева неуралне мреже.

```
def define_neurons(X, Y):
    Y = one_hot(Y)
    input_layer_neurons = X.shape[0]
    hidden_layer_neurons = 128
    output_layer_neurons = Y.shape[0]
    return (input_layer_neurons, hidden_layer_neurons, output_layer_neurons)
```

Слика 4.2 Функција која дефинише величине слојева неуралне мреже

#### 4.1.4 Модел *MNIST* неуралне мреже

Модел неуралне мреже се описује праговима и синаптичким тежинама између слојева. Наш модел укључује четири листе: *weights\_hidden* (тешине скривеног слоја), *biases\_hidden* (прагови скривеног слоја), *weights\_output* (тешине излазног слоја) и *biases\_output* (прагови излазног слоја). На слици 4.3 је приказана функција која описује почетни модел са насумично дефинисаним вредностима прагова и тежина између слојева неуралне мреже.

```
def initialize_parameters(input_layer_neurons, hidden_layer_neurons, output_layer_neurons):

    weights_hidden = [[random.uniform(-1, 1) * 0.1 for x in range(input_layer_neurons)] for x in range(hidden_layer_neurons)]
    biases_hidden = [[0.0] * 1 for x in range(hidden_layer_neurons)]

    weights_output = [[random.uniform(-1, 1) * 0.1 for x in range(hidden_layer_neurons)] for x in range(output_layer_neurons)]
    biases_output = [[0.0] * 1 for x in range(output_layer_neurons)]

    parameters = (weights_hidden, biases_hidden, weights_output, biases_output)

    return parameters
```

Слика 4.3 Функција која дефинише параметре почетног модела неуралне мреже

#### 4.1.5 Обучавање *MNIST* неуралне мреже

Обучавање неуралне мреже састоји се од више итерација које се извршавају у петљи. Свака итерација има следеће кораке:

- **Пропагација унапред (*forward\_propagation*):** Улазни подаци за обучавање (*X*) се пропагирају кроз мрежу користећи тренутне параметре модела неуралне мреже (*parameters*), што резултира активираним излазом (*activated\_output*) и излазима свих слојева мреже (*outputs*).
- **Пропагација уназад (*backward\_propagation*):** На основу излаза неуралне мреже и стварних ознака (*Y*), рачунају се градијенти помоћу пропагације уназад. Ови градијенти представљају меру грешке излаза у односу на стварне ознаке.

- **Ажурирање параметара:** Користећи градијенте из претходног корака, параметри неуралне мреже се ажурирају користећи алгоритам градијентног спуста (*gradient\_descent*). Циљ је минимизирати грешку неуралне мреже оптимизовањем параметара.

На крају свих итерација, функција *train* враћа ажуриране параметре неуралне мреже. На слици 4.4 је приказана функција која описује поступак обучавања *MNIST* неуралне мреже.

```
def train(X, Y, parameters, num_iterations=100):  
  
    for i in range(0, num_iterations):  
        activated_output, outputs = forward_propagation(X, parameters)  
        gradients = backward_propagation(parameters, outputs, X, Y)  
        parameters = gradient_descent(parameters, gradients)  
  
    return parameters
```

Слика 4.4 Функција која описује поступак обучавања *MNIST* неуралне мреже

#### 4.1.6 Тестирање *MNIST* неуралне мреже

Након што је неурална мрежа обучена, неопходно је спровести тестирање како бисмо проценили њену тачност и перформансе на подацима који нису коришћени током обуке. Улазни тестни подаци (*X\_test*) се пропагирају кроз мрежу користећи коначне параметре модела добијене након обучавања (*parameters*). Резултат ове пропагације су активирани излази (*activated\_output*) мреже. Из активираних излаза се одређују предвиђене вредности које се затим поређу са стварним ознакама (*Y\_test*) како би се израчунао број тачних предикција (*correct\_predictions*). Тачност модела се израчунава као проценат тачних предикција у односу на укупан број тестирањих узорака (*total\_digits*). На слици 4.5 се налази функција која описује поступак тестирања *MNIST* неуралне мреже.

```
def prediction(parameters, X_test, Y_test):  
    activated_output, outputs = forward_propagation(X_test, parameters)  
  
    correct_predictions = np.sum(np.argmax(activated_output, 0) == Y_test)  
    total_digits = Y_test.size  
    print("Correct Predictions: " + str(correct_predictions))  
    print("Total Digits Tested: " + str(total_digits))  
    print("Accuracy: " + str(round((correct_predictions / total_digits) * 100, 1)) + "%\n")
```

Слика 4.5 Функција која описује поступак тестирања *MNIST* неуралне мреже



## 4.2 Развој дистрибуиране апликације на окружењу *PTB-FLA*

У овом поглављу, детаљно ћемо описати кораке потребне за подешавање окружења, прилагођавање секвенцијалног кода за рад у *PTB-FLA* окружењу, као и саму имплементацију и покретање дистрибуиране апликације на окружењу *PTB-FLA*.

### 4.2.1 Подешавање окружења за рад

За ефикасно коришћење *PTB-FLA* окружења потребно је следити неколико корака како би се осигурало да су све потребне компоненте правилно конфигуриране. Прво, неопходно је преузети *ZIP* датотеку са *GitHub* репозиторијума [3]. У командној линији (*CMD*) креирати виртуелно окружење, инсталирати *PTB-FLA* заглавље и остале потребне библиотеке (у мом случају *numpy*). За писање и уређивање кода сам користио *Notepad++*, а за имплементацију *Python* верзију 3.12.0.

### 4.2.2 Четири фазе *PTB-FLA* развојне парадигме

*PTB-FLA* развојна парадигма омогућава постепену трансформацију секвенцијалног кода у *PTB-FLA* код кроз четири фазе. У свакој фази, код се прилагођава и оптимизује како би био у складу са специфичностима федеративног учења и окружењем у коме ће бити коришћен.

- **seq\_base\_case():** У првој фази, пише се основни секвенцијални код за *MNIST* неуралну мрежу. Овај код представља најједноставнији облик алгорита који обавља потребне операције без икаквих федеративних компоненти. Ова фаза служи као почетна тачка за даљу трансформацију кода.
- **seq\_horizontal\_federated():** У другој фази, основни секвенцијални код се трансформише у секвенцијални федеративни код. Овај корак подразумева измену кода како би се омогућило хоризонтално федеративно учење. То значи да се подаци и процес учења дистрибуирају на више клијената, али без додатних функција повратног позива.

- **seq\_horizontal\_federated\_with\_callbacks():** Трећа фаза уводи функције повратног позива у секвенцијални федеративни код. Ове функције омогућавају повратне информације током процеса учења, што побољшава комуникацију између клијената и сервера. Овај корак је кључан за имплементацију сложенијих алгоритама федеративног учења. Слика 4.6 описује клијентску и серверску функцију повратног позива.

```
def fl_cent_client_processing(parameters, privateData, srv_model):

    X_train = privateData[0]
    y_train = privateData[1]
    parameters = train(X_train, y_train, parameters)
    return parameters

def fl_cent_server_processing(privateData, models):

    weights_hidden_mean = 0
    biases_hidden_mean = 0
    weights_output_mean = 0
    biases_output_mean = 0

    num_clients = len(models)

    for model in models:
        model = ListToNumpy(model)
        weights_hidden_mean += model[0]
        biases_hidden_mean += model[1]
        weights_output_mean += model[2]
        biases_output_mean += model[3]

    parameters = (
        weights_hidden_mean / num_clients,
        biases_hidden_mean / num_clients,
        weights_output_mean / num_clients,
        biases_output_mean / num_clients
    )

    return NumpyToList(parameters)
```

*Слика 4.6 Клијентска и серверска функција повратног позива*

- **main():** У последњој фази, секвенцијални федеративни код са функцијама повратног позива се прилагођава за *PTB-FLA* окружење. Ова фаза подразумева финалну интеграцију свих компоненти и оптимизацију кода како би био потпуно функционалан у *PTB-FLA* систему.

### 4.2.3 Покретање дистрибуиране апликације на *PTB-FLA* окружењу

Покретање апликације на *PTB-FLA* окружењу захтева неколико корака. Након што се активира виртуелно окружење, потребно је позиционирати се у директоријум у ком се налазе примери имплементације. Апликација се затим покреће путем командне линије (*CMD*) коришћењем специфичних команди. На пример, за покретање апликације са три чвора, користи се команда `launch example7_NN_MNIST.py 3 id 2`, док се за покретање са четири чвора користи команда `launch example7_NN_MNIST.py 4 id 3`. Ове команде омогућавају извршавање примера за обучавање *MNIST* неуралне мреже у *PTB-FLA* окружењу, омогућавајући централизовано федеративно учење и тестирање на више чворова.

## 4.3 Развој дистрибуиране апликације на окружењу *MTB-FLA*

У овом поглављу, детаљно ћемо описати кораке потребне за подешавање окружења, прилагођавање кода *PTB-FLA* апликације за *MicroPython*, као и саму имплементацију и покретање дистрибуиране апликације на окружењу *MTB-FLA*.

### 4.3.1 Подешавање окружења за рад

За развој дистрибуиране апликације на окружењу *MTB-FLA*, прво је потребно правилно подесити окружење за рад. У овом случају, коришћена је *RPi Pico W* плочица на коју је убачен *firmware* који садржи *uLab* библиотеку [4], јер *MicroPython* не подржава *numpy* библиотеку која је коришћена до тада. Као интегрисано развојно окружење (*IDE*) коришћен је *Visual Studio Code*.

Поред тога, како би се апликација правилно покретала, у пројектни директоријум је потребно додати претходно преузети *PTB-FLA* директоријум са свим потребним фајловима. Ови кораци осигуравају да сви алати и библиотеке потребни за развој и имплементацију дистрибуираних алгоритама у *MTB-FLA* окружењу буду исправно постављени и доступни.

### 4.3.2 Прилагођавање *PTB-FLA* апликације за *MicroPython*

У *MicroPython* окружењу, главни проблем представља недостатак *numpy* библиотеке, па смо уместо ње користили *uLab* библиотеку као алтернативу. Како бисмо превазишли овај недостатак, морали смо прилагодити код како бисмо искористили функционалности *uLab* библиотеке.

Један од главних изазова био је рад са ограниченом радном меморијом доступном на *RPi Pico W* плочицама. Због ове ограничености, били смо приморани да смањимо величину улазних слика са оригиналних 28x28 пиксела на мање захтевних 9x9 пиксела. Ова оптимизација је била клучна за ефикасно управљање ресурсима.

Додатно, због немогућности спровођења обуке са великим скупом података директно на *RPi Pico W* плочицама, формирали смо систем где је почетно обучавање извршено на серверу са скупом од 1000 узорака. Након тога, клијенти на *RPi Pico W* плочицама су извршавали процес дообучавања са много мањим скупом од само 50 узорака, што је допринело бржем и ефикаснијем учењу система на самим уређајима.

Овај приступ не само да је омогућио ефикасније коришћење ресурса већ је и показао да су резултати добијени од клијената путем агрегирања били прецизнији у поређењу са онима добијеним приликом почетног обучавања на серверу.

#### 4.3.3 Покретање *MTB-FLA* апликације

За покретање *MTB-FLA* апликације, прво је потребно активирати виртуелно окружење и затим се позиционирати у директоријум где се налази имплементација апликације. Сервер покрећемо путем командне линије (*CMD*) коришћењем команде: **python mp\_async\_example7\_NN\_MNIST.py 3 0 0 192.168.160.12** Аргументи представљају име датотеке, број чворова, *ID* чвора, *ID* сервера и мастер *IP* адресу. Пре покретања клијената на *RPi Pico W* плочицама, неопходно је попунити конфигурациону датотеку (*config.py*) са тачним информацијама о *WiFi* мрежи на коју су уређаји повезани, као и са аргументима командне линије за сваког клијента. Пример конфигурационе датотеке можете видети на слици 4.7.

```
cfg = {
    "wlan": {
        "ssid": b"xxxx",
        "pswd": b"yyyy"
    },
    "myIP": "",

    # Test: node 0 in PC at IP '192.168.184.12' and nodes 1-2 in two rp2s with respective args below.
    #"argv": ['mp_async_example7_NN_MNIST', '3', '1', '0', '192.168.184.12'], # argv for the node 1 in rp2 no. 1
    "argv": ['mp_async_example7_NN_MNIST', '3', '2', '0', '192.168.184.12'], # argv for the node 2 in rp2 no. 2
}
```

Слика 4.7 Пример конфигурационе датотеке

## 5. Испитивање

У овом делу рада анализирамо резултате тестирања дистрибуиране *MTB-FLA* апликације. Тестирање је извршено на *PC* рачунару уз помоћ *RPi Pico W* плочице. Сервер и један клијент су покренути на мрежи преко *PC* рачунара, док је други клијент покренут на *RPi Pico W* плочици. Приказаћемо и дискутовати резултате, користећи табеле и графике за јаснију илустрацију закључака.

### 5.1 Резултати тестирања

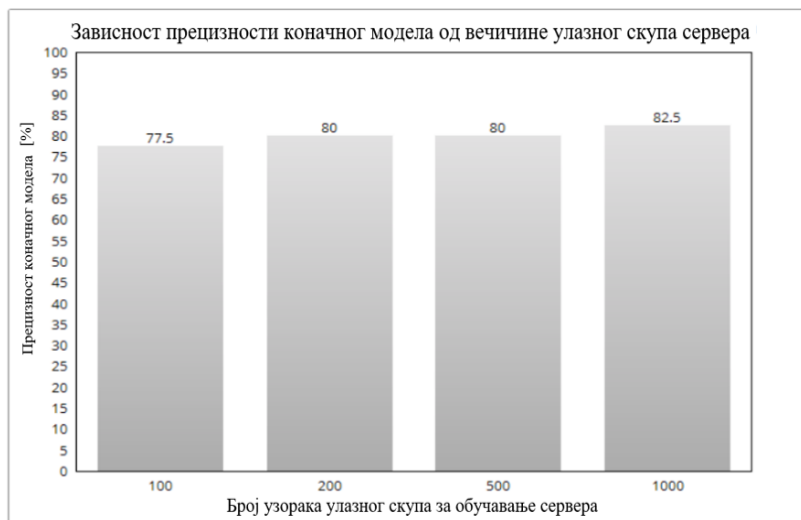
Фокусираћемо се на два кључна аспекта: утицај величине улазног скупа података и броја итерација сервера на прецизност крајњег модела. Величина улазног скупа података клијената је веома мала и нема смисла додатно је смањивати и испитивати утицај на прецизност крајњег модела. Такође, због малог улазног скупа података клијената, број итерација потребан за достизање доброг резултата је мали и испитивање за вредности веће од 300 нема пуно смисла.

#### 5.1.1 Зависност прецизности коначног модела од величине улазног скупа података сервера

Анализирали смо четири случаја за величине улазног скупа података сервера од 100, 200, 500 и 1000 узорака. Из табеле резултата 5.1 можемо да видимо подразумевање вредности осталих параметара и закључимо да са повећањем величине улазног скупа података имамо бољу прецизност крајњег модела. Такође, можемо видети да и мали улазни скуп података за ове вредности параметара даје солидне резултате. На слици 5.1 је приказан график зависности прецизности крајњег модела од величине улазног скупа података сервера.

Величина улазног скупа података сервера	Величина улазног скупа података клијената	Број итерација сервера	Број итерација клијената	Прецизност серверовог модела	Прецизност коначног модела
<b>100</b>	50	1000	300	65.0 %	77.5 %
<b>200</b>	50	1000	300	72,5 %	80,0 %
<b>500</b>	50	1000	300	72,5 %	80,0 %
<b>1000</b>	50	1000	300	77,5 %	82,5 %

*Табела 5.1 Зависност прецизности коначног модела од величине улазног скупа података сервера*



*Слика 5.1 График зависности прецизности коначног модела од величине улазног скупа података сервера*

### **5.1.2 Зависност прецизности коначног модела од броја итерација сервера**

У овом примеру смо такође имали четири случаја за вредности броја итерација 100, 500, 700 и 1000. Из табеле резултата 5.2 можемо да видимо вредности осталих параметара и закључимо да се прецизност крајњег модела повећава са повећањем броја итерација сервера. На слици 5.2 је приказан график зависности прецизности крајњег модела од броја итерација сервера.

Величина улазног скупа података сервера	Величина улазног скупа података клијената	Број итерација сервера	Број итерација клијената	Прецизност серверовог модела	Прецизност коначног модела
1000	50	<b>100</b>	300	22,5 %	40,0 %
1000	50	<b>500</b>	300	67,5 %	67,5 %
1000	50	<b>700</b>	300	67,5 %	77,5 %
1000	50	<b>1000</b>	300	77,5 %	82,5 %

*Табела 5.2 Зависности прецизности коначног модела од броја итерација сервера*



*Слика 5.2 График зависности прецизности коначног модела од броја итерација сервера*

## 6. Закључак

У овом раду смо детаљно анализирали и приказали развој дистрибуиране апликације за обучавање неуралне мреже коришћењем *MNIST* скупа података. Рад је фокусиран на употребу *PTB-FLA* и *MPT-FLA* окружења, омогућавајући прилагођавање апликација за рад на уређајима са ограниченим ресурсима као што је *RPi Pico W*.

Експерименти су показали да се прецизност крајњег модела повећава са већим улазним скупом података сервера и већим бројем итерација сервера. Кроз ова испитивања, показали смо да је могуће остварити задовољавајуће резултате чак и са ограниченим ресурсима, што отвара могућности за даљу примену на различитим *IoT* уређајима.

Додатно, дообучавање модела које је извршено на *RPi Pico W* плочицама са малим скупом података показало је добре резултате. Иако су ресурси на овим уређајима ограничени, успели смо да постигнемо солидну прецизност крајњег модела, што потврђује ефикасност и практичност овог приступа за реалне *IoT* апликације.

Иако су резултати сасвим задовољавајући, главно ограничење апликације на *MPT-FLA* су ограничени ресурси *RPi Pico W* плочице, због чега није могуће остварити тачност на нивоу који је остварив на *PC* рачунарима.

У будућем раду би свакако било интересантно истражити могућност рада *RPi Pico W* плочица на батеријском напајању и с тим у вези зависност потрошње електричне енергије од захтеване тачности модела. Други правац будућег рада би могао да буде развој других *ML* и *AI* дистрибуираних апликација применом исте развојне парадигме.



## 7. Литература

- [1] M. Popovic, M. Popovic, I. Kastelan, M. Djukic, and S. Ghilezan, *A Simple Python Testbed for Federated Learning Algorithms*, 2023, in: Proceedings of the 2023 Zooming Innovation in Consumer Technologies Conference, 2023, pp. 148-153, <https://doi.org/10.1109/ZINC58345.2023.10173859>.
- [2] M. Popovic, M. Popovic, I. Kastelan, M. Djukic, and I. Basicovic, “*MicroPython Testbed for Federated Learning Algorithms*”, 2024, doi: <https://doi.org/10.48550/arxiv.2405.09423>.
- [3] M. Popovic, *Github* репозиторијум *PTB-FLA* [Online]. Доступно на адреси: <https://github.com/miroslav-popovic/ptbfla>.
- [4] *Firmware* за *RPi Pico W* плочицу са укљученом *uLab* библиотеком. Доступно на адреси: <https://github.com/v923z/micropython-builder/releases>.