România
Ministerul Apărării Naționale
Academia Tehnică Militară "*Ferdinand I*"

Facultatea de Sisteme Informatice și Securitate Cibernetică
CALCULATOARE ŞI SISTEME INFORMATICE PENTRU APĂRARE ŞI SECURITATE NAŢIONALĂ

Platformă de Analiză Automată a Atacurilor Data Poisoning asupra unei Infrastructuri de Învățare Federate

Coordonator Științific
Cpt. conf. dr. ing. Iulian Aciobăniței

Absolvent
Sd. Sg. Maj. Lepădatu Tudor

Conține _____ file
Inventariat sub numărul _____
Cu poziția din indicator _____
Cu termen de păstrare _____

București
An 2026

Mulțumiri pentru persoanele care au sprijinit procesul de realizare a acestei lucrări

Referatul Coordonatorului Științific

Referatul reprezintă un text în care coordonatorul științific sumarizează, ulterior finalizării conținutului efectiv, efortul pe care l-ați depus și trage concluzii cu privire la gradul de realizare a obiectivelor propuse inițial (cele prezentate în cadrul detalierii). De regulă, acest referat nu depășește cele 4 pagini alocate în acest document.

Tema Proiectului de Diplomă

Tema proiectului de diplomă (sau detalierea) reprezintă un text realizat de coordonatorul științific, în lunile următoare propunerii titlului proiectului de diplomă către facultate, în care detaliază nevoia reală a implementării unui astfel de proiect și modul în care se dorește a fi implementat. În plus, poate prezenta structura pe capitole a viitoarei lucrări, anexele ce vor fi incluse și sursele bibliografice din care studentul se va informa. De regulă, acestă detaliere nu depășește cele 4 pagini alocate în acest document.

Abstract

The Artificial Intelligence integration with tools and applications has evolved since 2020s and the cybersecurity scene tries to adapt frequently. Information security and data integrity is more important than never before, being used by Machine Learning models or Neural Networks trained to perform specific tasks.

From a data science point of view, the quality of information is much important than securing it. The AI evolution has led to the creation of different attack boundaries, from changing the model parameters to perform data poisoning schemes. Confidentiality is the key in maintaining unique aspects for each entity involved in the federated learning process.

In this paper, data poisoning attacks are studied with different options in a segregated simulated infrastructure called federated learning, in which each client may change its scope intentionally or unintentionally. Each simulation has its own configuration providing data scientist with a dedicated environment for testing its machine learning algorithm againts data poisoning attacks.

Cuprins

Listă de figuri

Listă de Abrevieri

UE ..... Uniunea Europeană

EU ..... *European Union*

Capitolul 1:

Introducere

1.1 Context

Odata cu dezvoltarea sistemelor de calcul moderne si a componentelor Hardware, s-au putut realiza produse software complexe cu capacitati de stocare net superioare. Revolutia tehnologica a permis nu doar realizarea unor sarcini simple, precum calcule matematice, sau automatizarea unor dispozitive (de exemplu aprinderea automata a unui bec printr-un microcontroler), ci si posibilitatea gestionarii mai eficiente a informatiilor digitale (de la date bancare la fisiere media).

Aceasta a devenit treptat principala sursa legitima de inregistrare a oricarui tip de date (text, imagini, video, audio). Pentru a accesa si actualiza informatia digitala, s-au dezvoltat diferite versiuni de baze de date centralizate si distribuite.

Bazele de date centralizate sunt aplicatii software specializate ce folosesc resursele sistemului (a statiei) pentru a raspunde cat mai rapid interogarilor. Statiile trebuie sa detina multa putere de stocare si de procesare in comparatie cu un sistem de calcul normal destinat utilizatorilor casnici. In alta ordine de idei, s-au dezvoltat si baze de date distribuite, menite sa reduca din capacitatile tehnice ale serverului si sa stocheze informatia sub forma descentralizata. Cautarea resursei in acest context ar presupune interogarea recursiva a fiecarei entitati pana la gasirea sa. Prin acest mod, nu doar ca statiile pot avea si capabilitati tehnice mai reduse, dar si pot pastra copii de rezerva (backup) locale pentru fiecare segment de informatie in parte.

Această evoluție naturală spre descentralizare a deschis drumul unor concepte moderne precum învățarea automată federată (federated learning), unde datele nu mai sunt transferate către un server central. În schimb, modelele sunt antrenate local, iar parametrii sunt ulterior agregați global. Astfel, se menține confidențialitatea datelor, fără a compromite performanța modelului.

Evolutia tehnologica continua a dat nastere la o serie de atacuri cibernetice menite sa destabilizeze securitatea aplicatiilor si totodata sustragerea a cat mai multe date sau identitati private in contradictie cu normele legale. Cele mai populare atacuri raportate la scara globala pentru anul curent 2025 sunt ransomware (conform [1], in SUA s-au raportat cresteri de 149%), furtul de identitate prin exfiltrarea de credentiale, si phishing. Dezvoltarea modelelor de inteligență artificială a amplificat aceste riscuri, oferind atacatorilor instrumente automatizate pentru generarea și adaptarea atacurilor.

Pentru a limita utilizarea abuzivă a tehnologiilor bazate pe AI, Uniunea Europeană a adoptat în 2024 un set de reglementări stricte privind integrarea acestor module în aplicațiile software, prin AI Act [2].

Progresul din domeniul machine learning si a Large Language Models a fost posibil ca urmare a unui volum masiv de date disponibile si a nevoii tot mai mari de analiza. Acest lucru a determinat aparitia unei noi categorii de specialisti, data scientists, dedicati colectarii si prelucrarii minutioase a datelor pentru antrenarea modelelor.

Totuși, pe măsură ce investițiile în tehnologii AI au crescut, au apărut și actori rău intenționați care încearcă să exploateze vulnerabilitățile din procesul de antrenare. Întrucât modelele moderne depind de calitatea datelor folosite, acestea au devenit o țintă principală a atacurilor. Atacatorii se regasesc si ei intr-o pozitie constanta de adaptare la noile formalitati de securitate si incearca sa contracareze fiecare element nou. Astfel, avand in vedere complexitatea dezvoltarii unui modul de inteligenta artificiala specializat pe diferite domenii, tinta s-a redirectionat spre volumul de date pe care acestea le folosesc si care pot determina starea finala a aplicatiei.

În contextul învățării automate distribuite, literatura de specialitate identifică trei categorii majore de atacuri:

- Atacuri asupra datelor, precum data poisoning, unde setul de antrenare este manipulat pentru a altera comportamentul modelului;

[1] Directoratul Național de Securitate Cibernetică (DNSC), *Buletin de Indicatori Statistici și Tendințe de Securitate Cibernetică - H1 2025*, `https://www.dnsc.ro/vezi/document/buletin-de-indicatori-statistici-si-tendinte-de-securitate-cibernetica-h1-2025`, Accesat în 2025-11-27, 2025

[2] Future of Life Institute, *AI Act Overview - Future of Life Institute (30 May 2024)*, `https://artificialintelligenceact.eu/wp-content/uploads/2024/11/Future-of-Life-InstituteAI-Act-overview-30-May-2024.pdf`, Accessed: 2025-11-27, mai 2024

- Atacuri asupra modelului, prin modificarea parametrilor sau a gradientului (de exemplu, model poisoning);

- Atacuri asupra canalului de comunicare, care vizează interceptarea sau modificarea mesajelor dintre entitățile participante.

Lucrarea de față se concentrează pe prima categorie, data poisoning, în cadrul unei infrastructuri de invatare federate.

1.2 Motivatia lucrarii

Avand in vedere aspectele legate de posibilitatea unei interventii asupra setului de date de antrenare, atac denumit otravire a datelor (data poisoning), munca cercetatorilor s-a ingreunat. Preocuparea nu mai este primordial asupra analizei setului de date de antrenare, cat despre mentinerea integritatii si a confidentialitatii lor. Pentru a răspunde acestor nevoi, colaborarea dintre cercetători s-a orientat către modele distribuite de lucru, iar învățarea federată (federated learning) a devenit una dintre principalele direcții. Aceasta permite colaborarea între participanți fără a partaja direct seturile lor de date, menținând o barieră naturală împotriva accesului neautorizat. Totuși, deși infrastructura este diferită față de abordările centralizate, vulnerabilitățile rămân, iar atacurile asupra datelor utilizate local pot afecta modelul global.

In urma unei analize proprii, am putut observa diferite solutii/frameworks de simulare a procesului de invatare automata federata, dar fara o integrare cu mecanisme moderne de testare pentru atacuri precum otravirea datelor (data poisoning) amintite anterior [3]. Unele dintre aceste framework-uri sunt poate dificil de gestionat si configurat [4], si nu permit extinderea usoara prin integrare altor componente. In acelasi timp, gandindu-ne la multitudinea de atacuri malware si la platformele de detectie a lor, devine clar ca in domeniul inteligentei artificiale lipseste o platforma centralizata, flexibila, dedicata testarii si evaluarii cu diferite tipuri de atacuri asupra modelelor distribuite.

Aceste limitări justifica realizarea prezentei lucrari, care își propune dezvoltarea unei platforme de simulare capabile sa testeze atacuri de tip data poisoning într-o infrastructura de învățare federată.

1.3 Obiectivele lucrarii

Plecand de la neajunsurile prezentate, ne propunem in aceasta lucrare sa venim in sprijinul comunitatii de cercetare stiintifica in domeniul securizarii solutiilor cu AI cu o platforma de simulare cu sursa deschisa ("open source"), a acestei clase de atacuri pe mai multe directii. Astfel, oferim cercetatorilor posibilitatea analizei algoritmului de antrenare propriu dezvoltat, plecand de la o retea neuronala de baza si un set de date uzual (imagini), si testarea sa prin antrenare in diferite conditii. Platforma in sine respecta toate normele unei aplicatii software de productie, in care fiecare actiune are propria sa logica de implementare. Serviciile sunt segregate suficient de mult incat sa permita o dezvoltare ulterioara prin integrarea lor cu alte sisteme.

Rezultatele pot fi utile in contextul securizarii procesului de antrenare al algoritmului, dar si pentru analiza factorilor de risc la care e expus in acest mediu.

Cercetatorul este cel care furnizeaza algoritmul python de antrenare a propriei retele neuronale sau algoritm de Machine Learning. El seteaza parametrii simularii atat pentru procesul de antrenare, cat si pentru tipul de atac de otravire a datelor. Platforma isi propune sa simuleze acest tip de atac cu ajutorul acestor setari de inceput intr-un mediu de invatare federata, furnizand la final o comparatie intre modelul antrenat folosind datele normale de antrenare si cel antrenat cu datele otravite. Aceste rezultate pot fi utile in semnalarea unui posibil risc la nivelul modelului dezvoltat, oferind mai apoi solutii de imbunatatire a implementarii sale.

1.4 Structura lucrarii

[3] IBM, *IBM Federated Learning API Documentation*, `https://ibmfl-api-docs.res.ibm.com/index.html`, Accesat în 2025-11-28, 2025

[4] IBM, *IBM Federated Learning Library - GitHub Repository*, `https://github.com/IBM/federated-learning-lib/tree/main`, Accesat în 2025-11-28, 2025

Capitolul 2:

Notiuni Teoretice

In acest capitol, vor fi prezentate notiunile teoretice specifice intelegerii procesului de dezvoltare a platformei de simulare. Vom incepe cu Notiunile introductive despre conceptele de Machine Learning in antiteza cu Deep Learning. In continuare, vom discuta despre invatarea federata si arhitectura unei infrastructuri federate de invatare automata, tipurile de atacuri data poisoning implementate in procesul de simulare a atacurilor, precum si alte notiuni specifice implementarii.

2.1 Notiuni introductive

Machine Learning si Deep Learning sunt doua ramuri importante ale Inteligentei Artificiale care au rolul dezvoltarii unor modele specifice rezolvarii unor anumite actiuni. Pornind de la antrenarea de retele neuronale, ne orientam atentia spre setul de date de antrenare si spre actorii ce pot interveni in acest proces. Mediul in care testam ofera o perspectiva reala asupra impactului pe care il pot avea aceste atacuri la nivelul unei organizatii sau aplicatii.

2.1.1 Diferenta dintre Machine Learning si Deep Learning

Inteligența Artificială (AI) este domeniul vast care înglobează orice tehnică ce permite calculatoarelor să imite comportamentul uman. Informatia a evoluat treptat odata cu imbunatatirea capabilitatilor de stocare ale dispozitivelor si aparitia programelor software complexe. De la simplul format de text, inregistrari audio, pana la imagini si video in rezolutii 4K, modul de lucru s-a diversificat constant.

La fel au evoluat si cerintele utilizatorilor, care tind sa acceada catre solutii automate care sa le rezolve problemele uzuale, precum identificarea de patterns in imagini sau chiar din video, sau generarea de text.

IA vine sa rezolve aceste probleme si sa introduca algoritmi de rezolvare specifici pentru fiecare tip de informatie furnizata.

Machine Learning este o componenta importanta din domeniul IA care se diferentiaza de alte metode de antrenare prin optimizarile pe care le aduce erorilor ce apar din predictia rezultatului corect. Modelele de ML clasice se bazeaza pe interventia umana in factorul de decizie (supervised learning), mai precis datele de intrare sunt etichetate pentru a oferi un context de predictie stabil.

Deep Learning este o subcategorie a Machine Learning, care are rolul de a minimiza interventia umana si a automatiza procesul de decizie. Prin aceasta metoda se automatizeaza mare parte din extragerea caracteristicilor pe setul de date, eliminand nevoia de a defini etichete pentru fiecare valoare de intrare (unsupervied learning).

Diferenta dintre aceste doua concepte este in modul in care acesti algoritmi invata si procentul de utilizare a datelor [1]. Scopul principal al invatarii automate este predictia. Pe baza unui set de date de antrenare si de testare, se determina o anumita categorie de iesire predefinita.

2.1.2 Retea Neuronala

Retelele Neuronale sunt un subset al Machine Learning si se identifica drept infrastructura de baza din cadrul algoritmilor de Deep Learning. Denumirea de "neuronal" se refera la structura lor interna, in care fiecare caracteristica (feature) este un neuron ce interactioneaza unii cu altii. Ele sunt compuse din 3 straturi/layers: primul strat il reprezinta stratul nodurilor de intrare, al doilea strat este denumit "stratul ascuns" (hidden layer) pt ca incapsuleaza mai multe straturi, iar ultimul strat este cel de iesire in care se face predictia propriu-zisa. Straturile ascunse sunt concepute pentru a procesa iterativ datele pornind de la starea lor din nodurile de intrare pana la stratul de iesire.

[1] IBM, *AI vs. Machine Learning vs. Deep Learning vs. Neural Networks*, `https://www.ibm.com/think/topics/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks`, Accesat în 2025-11-28, 2025

2.2 Invatare automata federata

Evolutia hardware in tehnologie a condus la cresterea numarului dispozitivelor mobile (telefoane, tablete), denumite gadgets din faptul ca sunt mici, portabile si moderne. Ele au fost mai departe adoptate la scara larga, devenind obiecte indispensabile in era tehnologica ce avea sa vina.

2.2.1 Concept

Invatarea automata federata permite lucrul cu modele de ML sau chiar retele neuronale, antrenate distribuit, pe un numar mare de dispozitive in scopul rezolvarii unei probleme de IA. Distribuirea sarcinilor a fost adoptata si in contextul opozitiei lucrului centralizat, pe servere ce detin capabilitati Hardware performante (placi grafice de ultima generatie), dar care genereaza costuri mari si care pot fi predispuse la amenintari de securitate cibernetica, fiind considerate SPOF(Single Point of Failure).

2.2.2 Arhitectura FL

In literatura, exista mai multe categorii de arhitecturi de invatare automata federata. In aceasta sectiune ne vom concentra pe clasificarea generala a arhitecturii unei aplicatii folosind federated learning, si vom enumera pentru o anumita categorie cum se clasifica dispozitivele utilizate.

Federated Learning, asa cum a mai fost mentionat, este organizat dintr-un server (agregator) si multiple dispozitive client. Modul in care aceste entitati comunica este fundamentul principal in modului de imbunatatire al invatarii.

In modul clasic al federated learning, dispozitivele client transmit actualizari ale modelului de baza la un server central care aplica asupra lor o functie de agregare, reconstruind intreg modelul de baza. Aceasta setare/model, presupune de fapt o delegare a sarcinii de invatare, insa pastreaza entitate centrala necesara imbunatatirii solutiei. Acest fapt, nu tine sa evita posibilitatea amenintarilor cibernetice (Single Point of Failure), ci doar sa usureze costurile centralizatorului in a procesa local problema, distribuind sarcinile.

Modelul Fully decentralized (peer-to-peer) learning, ofera o noua abordare si rezolva problema cibernetica amintita. In aceasta setare nu exista agregator, imbunatatirile fiind comunicate intre clienti interconectati. Ideea principala se bazeaza pe inlocuirea comunicarii cu agregatorul cu cea intre dispozitive individuale printr-un protocol prestabilit. In functie de numarul de dispozitive, se concepe un graf de conexiuni in care fiecare nod reprezinta un client, iar fiecare muchie un canal de comunicatie. Restrictia principala este ca un dispozitiv sa fie conectat la un numar maxim limitat de dispozitive adiacente, prestabilit, in contradictie cu un graf complet (stea) specific arhitecturii clasice client-server. Nodurile isi imbunatatesc propriile variante ale retelei, si isi comunica rezultatele pe care le agrega local, realizand o medie a ponderilor. In comparatie cu modelul federated learning clasic, modelul fully decentralized nu specifica de la inceput dispozitivelor un model de baza global de la care sa porneasca in procesul de rezolvare a problemei.

	Federated Learning	**Fully Decentralized (peer-to-peer) Learning**
Orchestrare	Un server central de orchestrare organizează antrenarea, dar nu vede niciodată datele brute.	Nu există orchestrare centralizată.
Comunicare pe arie largă	De obicei topologie hub-and-spoke, în care hub-ul reprezintă un furnizor de servicii coordonator (de obicei fără date), iar spoke-urile conectează clienții.	Topologie peer-to-peer, cu un graf de conectivitate posibil dinamic.

Tabel 2.1: Comparație între modelele Federated Learning [10]

Imaginea de mai sus ofera o privire de ansamblu asupra celor doua modele de arhitecturi si caracteristicile acestora.

Model	Avantaje	Dezavantaje
Federated learning (centralized coordination)	• mai simplu de configurat (topologie hub-and-spoke) • pornește de la un model global de bază • agregarea centralizată reduce sarcina de calcul pe clienți • necesită mai puține conexiuni (doar client → server) • gestiunea și monitorizarea sunt mai simple	• SPOF (Single Point of Failure) - serverul central • serverul poate deveni țintă pentru atacuri • nu elimină riscurile cibernetice, doar distribuie munca • dependența de coordonator pentru progresul antrenării • necesită infrastructură centralizată permanent disponibilă
Fully decentralized / peer-to-peer learning	• previne atacurile specifice unui server central (evită SPOF) • reziliență crescută - compromiterea unui nod nu destabilizează întregul sistem • îmbunătățirile se propagă prin graf, fără entitate centrală • agregare locală (fiecare nod mediază ponderile) • poate scala natural dacă graful este bine proiectat	• necesită conexiuni suplimentare între clienți • topologie complexă, dificil de administrat • nu există model global inițial furnizat tuturor • performanța depinde de calitatea grafului de conexiuni • nodurile malițioase pot influența direct vecinii • necesită protocoale suplimentare pentru consistența actualizărilor

Tabel 2.2: Compararea modelelor Federated Learning și Fully Decentralized Learning

In tabelul de mai sus, sunt evidentiate avantajele si dezavantajele utilizarii celor doua tipuri de modele federated learning.

In continuarea acestei lucrari, se va discuta preponderent despre modelul clasic federated learning, fiind unul adoptat la scara larga si care ofera performante bune raportat la costurile de productie.

Modelul clasic la randul sau, se catalogheaza in literatura in functie de tipul dispozitivelor care iau parte la procesul de antrenare. Sub acest filtru, exista Cross-Device Federated Learning si Cross-Silo Federated Learning.

Cross-Device Federated Learning sunt dispozitivele client IOT uzuale, individuale, care comunica orchestratorului printr-un protocol prestabilit.

Cross-Silo Federated Learning sunt dispozitive din institutii guvernamentale, companii, sau centre de date distribuite geografic. Institutiile nu doresc sa schimbe informatii intre ele sau cu un furnizor de servicii central, pastrandu-si confidentialitatea, folosind federated learning pt a antrena propriul model pe datele private ale fiecaruia.

2.2.3 Procesul de antrenare FL

Primul pas este stabilirea conexiunii dintre dispozitive si un server de agregare ce permite antrenare distribuita a tipului de retea neuronala sau model ML specific problemei. Odata stabilit canalul, in faza de configurare initiala, serverul trimite dispozitivelor starea de baza a retelei neuronale, ponderile, in vederea antrenarii individuale. Fiecare retea se antreneaza cu datele extrase local (on device) si isi imbunatateste configuratia interna la fiecare epoca pentru o perioada de timp bine determinata.

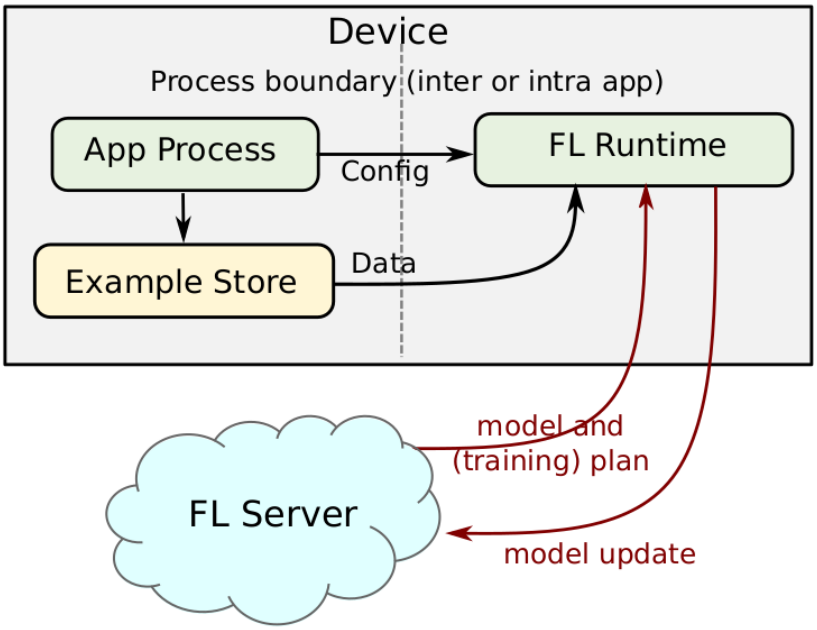

Figura 2.1: Arhitectura interna a unui dispozitiv [2]

Figura de mai sus descrie operatiile specifice programului Software care se ocupa de antrenarea retelei/modelului. Putem observa cum dispozitivele primesc un plan de antrenare de baza pe care il vor antrena local pe un set de date limitat.

Desi acest pas nu aduce un procent de imbunatatiri foarte mari, in faza urmatoare, dispozitivele vor transmite configuratiile curente ale retelelor lor la orchestrator (server). Rutina FL_Runtime extrage configuratia noua locala si ii comunica serverului pentru o posibila actualizare a sa.

In cele din urma, entitatea centrala combina toate aceste ponderi aplicand o functie de agregare si in cazul imbunatatirii setului de ponderi, modifica configuratia de baza si o retransmite dispozitivelor pereche. Daca ponderile noi nu se imbunatatesc semnificativ fata de configuratia de baza, atunci se patreaza aceasta din urma, iar in caz contrar se actualizeaza cu noile ponderi.

In figura de mai sus, se pot observa intr-o maniera continua, fluxul de comunicatie dintre dispozitive si serverul agregator, precum si operatiile specifice fiecarei entitati dintr-o runda de mesaje.

Securitatea protocoalelor de agregare, utilizate in comunicatii dintre clienti si orchestrator, este o componenta importanta in procesul federated learning. De mentionat este faptul ca, in aceasta topologie,

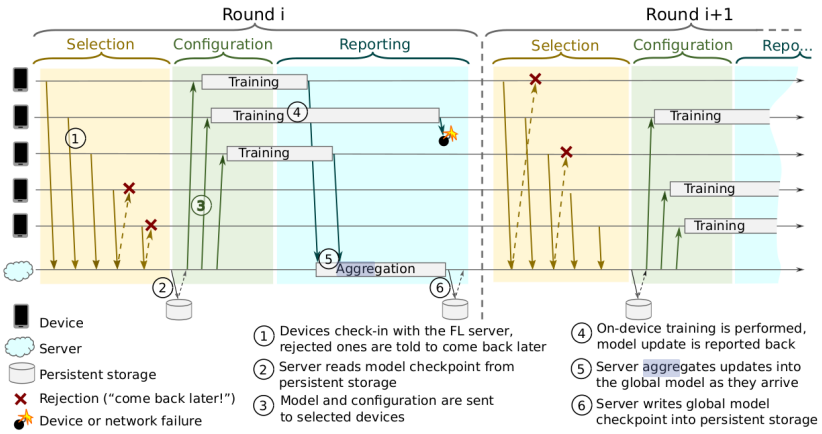

Figura 2.2: Procedee in invatare automata federata [3]

comunicatiile au loc criptat, folosind metode specifice precum criptare homomorfica, sau chiar OTP, insa securitatea datelor de pe dispozitive ramane la latitudinea acestuia.

2.2.4 Exemple in viata reala

Federated Learning s-a extins rapid în numeroase domenii datorită capacității sale de a antrena modele performante fără a colecta sau centraliza date sensibile. Prin păstrarea informațiilor la nivelul fiecărui dispozitiv sau instituții, FL reduce riscurile asociate scurgerilor de date și permite colaborarea între entități care altfel nu ar putea împărtăși date brute. În continuare sunt prezentate câteva exemple reprezentative ale utilizării sale în aplicații din lumea reală.

Industrie și IoT

- **Mentenanță predictivă:** Vehiculele moderne, utilajele industriale și echipamentele IoT generează constant date despre starea componentelor. FL permite antrenarea unui model comun care poate prezice momentul oportun pentru realizarea mentenanței fără a colecta date brute de la fiecare dispozitiv.
- **Dispozitive de monitorizare:** Senzori purtabili și dispozitive smart home pot furniza statistici privind activitatea sau consumul energetic, păstrând datele utilizatorilor la sursă.

Medical

- **Diagnostic, prognoză și imagistică:** FL este folosit în spitale și clinici pentru detectarea celulelor canceroase din imagini RMN, CT sau radiografii, fără transferul imaginilor către un server central.
- **Confidențialitate menținută la sursă:** Fiecare instituție medicală antrenează local o parte din model, partajând doar actualizările, ceea ce permite colaborarea fără a încălca regulile privind datele pacienților.

Financiar

- **Detectarea fraudelor:** Instituțiile financiare pot îmbunătăți detectarea tranzacțiilor suspecte analizând tipare comune fără a expune date sensibile despre clienți.

Servicii și experiență utilizator

- **Recomandări personalizate:** Platformele de streaming și aplicațiile mobile generează recomandări local, pe dispozitiv, fără a trimite istoricul complet al utilizatorului către server.
- **Analiză comportamentală:** FL poate analiza activitatea utilizatorilor pentru a sugera rutine sănătoase sau îmbunătățiri ale stilului de viață, păstrând confidențialitatea datelor.

Securitate și privacy

- **Supraveghere fără expunerea datelor sensibile:** Modelele de recunoaștere facială pot fi antrenate fără a transmite imagini reale, doar parametrii aferenți.
- **Analiză a sentimentelor:** FL poate analiza reacțiile utilizatorilor la evenimente sociale (like-uri, share-uri, comentarii) fără colectarea directă a acestor date de către platformă.

2.3 Atacuri de tip Data Poisoning

In acest capitol se va discuta una dintre avantajele pe care le ofera mediul federated learning atacatorilor si ce inseamna acest lucru pentru fluxul configuratiei modelului. Vom incepe cu definirea vectorilor de atac si concentrarea pe una dintre categorii, data poisoning. In continuarea lucrarii, vom analiza impactul pe care il are acest tip de atac asupra infrastructurii federate de invatare si riscurile pe care le introduce, precum si cateva propuneri de identificare si constientizare a existentei sale.

2.3.1 Definirea tipurilor de atac

In literatura de specialitate, atacurile asupra unui model de ML sau asupra unei retele neuronale sunt definite drept atacuri adversariale (adversarial attacks). Aceasta clasa de atacuri are rolul de a produce modificari in comportamentul normal al modelului intr-un mod indizerabil. Conform lucrarii [4], in functie de nivelul de scop al unui atac, putem avea:

- **Atacuri fără țintă (untargeted attacks)** Scopul este reducerea acurateții generale a modelului sau chiar destabilizarea completă a acestuia. Un exemplu pentru clasificarea imaginilor este introducerea unui zgomot care degradează calitatea setului de date. O altă metodă este modificarea etichetelor din setul de antrenare, de exemplu atribuirea etichetei *leu* unor imagini care în mod normal ar trebui etichetate drept *pisică*.
- **Atacuri țintite (targeted attacks)** Denumite și *backdoor attacks*, deoarece urmăresc modificarea comportamentului modelului doar pentru un anumit subset de date, fără a afecta vizibil acuratețea globală. Continuând exemplul anterior, pentru un anumit tip de imagini ce reprezintă pisici într-o anumită poziție, se poate introduce un artefact vizual menit să păcălească modelul și să clasifice pisica drept *leu*. Astfel apare un backdoor activ doar pentru acele imagini precise.

Plecand de la aceasta categorisire, exista o multime de modalitati prin care un atacator poate submina capacitatea de predictie a modelului. Mediul federated learning introduce prin constructie o serie de intrebari la care trebuie gasit un raspuns pentru a determina prin ce moduri un adversar se poate infiltra si poate profita de anumite drepturi pentru a introduce incertitudine in antrenarea sau reglarea (fine-tuning) a modelelor.

2.3.2 Vectori de atac

Analiza amenintarilor asupra modelelor de ML, a introdus o serie de posibile vulnerabilitati asupra componentelor ce alcatuiesc fluxul de invatare. Un atacator isi poate alege zona de interes, pe baza posibilitatilor de exploatare a sistemului respectiv.

Vectorii de atac cunoscuti sunt:

- **Data Poisoning** Cand adversarul incearca sa corupa setul de date antrenare cu scopul defectarii modelului inca de la inceput.
- **Model Update Poisoning** Cand adversarul se foloseste de o vulnerabilitate ce ii permite modificarea configuratiei parametrilor trimisi catre orchestrator.
- **Evasion attack** Cand adversarul are acces la datele de testare si le poate modifica in momentul inferentei.

In functie de gradul de acces la sistemele gazda, modelul poate fi inspectat in diferite moduri:

[4] Peter Kairouz et al., "Advances and Open Problems in Federated Learning", în *Foundations and Trends® in Machine Learning* 14.1-2 (2021), pp. 1–210, DOI: 10.1561/2200000083, URL: https://doi.org/10.1561/2200000083

- **Black Box** Adversarul nu are abilitatea sa inspecteze parametrii modelului inainte sau in timpul atacului.
- **Stale Box** Adversarul poate inspecta doar o versiune incipienta a modelului. Aceasta capabilitate apare si in federated learning cand adversarul are acces la rundele de antrenare ale clientului.
- **White Box** Adversarul are abilitatea de a inspecta direct parametrii modelului. Acest scenariu se bazeaza pe un grad de acces superior al adversarului asupra sistemului.

Pe baza acestor scenarii, atacatorul poate aplica o serie de tehnici pentru modificarea starii modelului. In contextul lucrarii de fata, se va discuta scenariul stale box, adversarul avand posibilitatea doar de a introduce un atac Data Poisoning in rundele de antrenare, datele corupte fiind introduse o singura data in procesul de reglare a modelului (fine-tuning).

2.3.3 Atacul Data poisoning

Acest tip de atac presupune coruperea datelor de antrenare sau de testare, prin diferite tehnici specifice, la nivelul dispozitivului clientului. In aceasta paradigma, atacul poate fi considerat la fel de bine targeted sau untargeted intrucat depinde de intentia adversarului si de potentialul risc in divulgarea punctului de exploatare de care dispune.

În practica atacurilor de tip Data Poisoning, adversarul poate manipula atât conținutul datelor, cât și etichetele acestora, efectele fiind de obicei greu de observat la nivel local. În mediul federated learning această dificultate este amplificată, deoarece orchestratorul nu are acces direct la datele brute ale clienților. Astfel, orice modificare realizată pe un dispozitiv compromis intră automat în procesul de antrenare, fiind tratată ca o contribuție legitimă. În mod particular, chiar și un număr redus de exemple otrăvite poate introduce un comportament persistent în modelul global, mai ales dacă atacul este repetat pe durata mai multor runde

In contextul solutiei propuse, se va discuta despre impactul atacului asupra unui set de imagini si tipurile de metode pentru a le altera, asa cum se poate vedea in tabelul 2.3.3.

Tip atac	**Descriere**
Gaussian noise	Introducerea unui zgomot aleator în imagini sau în vectorii de caracteristici, cu scopul degradării calității datelor și scăderii performanței modelului.
Label flip	Modificarea intenționată a etichetelor din setul de antrenare, astfel încât exemple corecte sunt asociate cu clase greșite, afectând procesul de învățare.
Backdoor injection	Inserarea unui artefact vizual sau a unui tipar specific într-un subset mic de date, astfel încât modelul să învețe un comportament anormal activat doar de acel trigger.

Tabel 2.3: Tipuri de atacuri Data Poisoning

Prin aceste tipuri de atacuri adversariale, impactul asupra modelului are loc pe o perioada determinata de timp, de obicei mai lunga, si produce variatii in predictia finala.

2.3.4 Impactul poisoning in Federated Learning

Mediul de invatare federata a introdus o serie de amenintari cibernetice preponderent la nivelul dispozitivelor clientilor, acestea fiind cele mai vulnerabile din punctul de vedere al aplicarii unui atac de otravire a datelor. Clientii sunt producatorii unui model calitativ care sa ofere predictii legitime in diferite scenarii.

Cand se discuta despre alterarea etichetelor (Label Flip Attack) atunci la o prima vedere, utilizatorul nu si-ar da seama decat in urma unei inspectii amanuntite. Pentru acest tip de atac, exista tehnici de verificare si filtrare ce pot determina daca un tip de informatie este catalogata corect inainte de antrenare.

Efectele unui data poisoning pot persista chiar și după eliminarea datelor corupte, deoarece modelul învață un tipar greșit care nu dispare imediat fără o reantrenare completă. Acest lucru este relevant mai ales pentru atacurile backdoor, care rămân inactive până la apariția unui trigger vizual, fără a afecta acuratețea generală. Din acest considerent, atacurile tintite (targeted attacks) sunt deosebit de periculoase in contextul unui mediu de invatare federata pentru ca datele corupte se ascund in interiorul configuratiilor particulare ale clientilor. Aceste configuratii sunt transmise mai departe la agregator care aplicand functia sa de agregare, amplifica negativ starea modelului.

Având în vedere aceste particularități, devine esențială analizarea metodelor de apărare și a mecanismelor prin care pot fi detectate contribuțiile malițioase. Provocarile in acest domeniu conduc la o serie tot mai mare de utilitare sau platforme ce permit detectarea facila a acestor atacuri si imbunatatirea evenimentelor cu un potential risc in organizatii.

Mediul federated learning introduce un risc suplimentar: un adversar care controlează un număr mic de clienți poate influența disproporționat modelul global dacă este integrat într-un moment critic al antrenării. În absența unor mecanisme robuste de apărare, actualizările malițioase sunt tratate ca fiind legitime, iar agregatorul nu are nicio modalitate directă de a le verifica.

Un alt efect important al acestui tip de atac este degradarea treptată a performanței modelului. În scenariile untargeted, scăderea acurateții globale poate trece neobservată în primele runde de antrenare, dar devine evidentă odată ce modelul converge către o reprezentare eronată a datelor. În scenariile targeted, atacul poate compromite decizii critice doar într-un subset de cazuri, ceea ce face detectarea mult mai dificilă și impactul mult mai nociv, mai ales în aplicații sensibile cum ar fi securitatea, domeniul medical sau sistemele autonome.

2.4 Alte Notiuni

In vederea elaborarii solutiei propuse in aceasta lucrare, se vor aminti celelalte concepte care stau la baza implementarii propriu-zise. In acest capitol se vor detalia succint mecanismele ce stau la baza platformei propuse, modul de utilizare si scopul alegerii lor.

2.4.1 Docker

Docker reprezintă un set de servicii software de tip platformă ce utilizează virtualizarea la nivel de sistem de operare pentru a crea entități independente, numite containere. Aceste containere sunt create specific pentru a întreprinde anumite acțiuni și oferă un mediu izolat de execuție.

Un container este o instanta software ce vine impachetata cu programul aplicatiei si toate bibliotecile necesare dezvoltarii ei. O imagine este vizualizata drept un sablon de instructiuni pentru crearea unui container cu un anumit tip de biblioteci necesare dezvoltarii unei aplicatii. De obicei, imaginile sunt construite pe baza unui fisier denumit Dockerfile care propune o serie de comenzi pentru crearea unui mediu personalizat.

Toate containerle Docker ruleaza prin intermediul Docker Engine, un serviciu ce ruleaza la nivelul sistemului de operare si ofera suport cross-platform (Linux, Windows, macOS).

Containerizarea diferă de virtualizarea tradițională prin faptul că containerele partajează același kernel al sistemului de operare gazdă, în timp ce mașinile virtuale (VM) necesită fiecare un sistem de operare complet. Această abordare face containerele Docker mult mai ușoare (de ordinul MB vs GB) și mai rapide la pornire (secunde vs minute) comparativ cu VM-urile.

Arhitectura platformei docker este asemanatoare modelului client-server, fiind compusa din urmatoarele componente:

- **Dockerd** un proces daemon, identificabil drept server, ce gestioneaza tot fluxul de servicii, de la imagini si containere pana la volume si retele.
- **Aplication Binary Interfaces (API)** o suita de interfete de comunicare si control al serverului
- **Comanda docker** o interfata in linie de comanda, docker

În contextul platformei dezvoltate, Docker ofera un mediu de dezvoltare automat in care se regaseste gazduita platforma publica de simulare. Toate informatiile despre fiecare simulare sunt stocate la nivelul unei baze de date PostgreSQL, in timp ce aplicatia web este publica printr-un container frontend, iar in spate regasim un container backend pentru transmiterea de comenzi serverului.

Comunicatia dintre containere are loc in aceeasi retea locala docker, iar informatiile despre fiecare simulare a clientului persista in acelasi volum partajat.

Pe langa Dockerfile, serviciul Docker Engine ofera si posibilitatea crearii unei configuratii prestabilite pentru definirea de topologii de retea de containere prin Docker Compose. Pe baza fisierelor Dockerfile in care sunt definite sabloanele imaginilor si a unui fisier de configurare YAML a topologiei relatiilor dintre containere, serviciul Docker Compose permite implementarea unei infrastructuri intregi prin rularea si stergerea sa dintr-o serie de comenzi.

2.4.2 Python

Python este unul dintre limbajele de programare cele mai des utilizate in contextul global actual. Python este un limbaj interpretat, de nivel inalt, ce vine cu o suita de biblioteci si functii utile in mai toate domeniile.

Intreaga implementarea operatiunilor backend din proiectul propriu-zis a fost redactata folosind limbajul python, fiind nu doar util pentru a programa aplicatii software, dar intalnit in Machine Learning, pentru a antrena retele neuronale si modele de invatare automata.

2.4.2.1 Biblioteci Python utilizate

Python a devenit de facto standardul in domeniul ML si Deep Learning datorita ecosistemului sau bogat de biblioteci specializate. In contextul inteligentei artificiale, cele mai intalnite biblioteci utilizate in platforma lucrarii sunt:

- **TensorFlow**: una dintre cele mai populare biblioteci open-source pentru Machine Learning, dezvoltată de Google Brain Team. TensorFlow oferă o platformă completă pentru construirea și antrenarea modelelor de Deep Learning, cu suport nativ pentru GPU, ceea ce accelerează semnificativ procesul de antrenare. GPU sunt placi grafice dedicate procesarii unui singur set de instructiuni simultan pe multiple seturi de date (SIMD), avand integrate mai multe coruri de procesare decat un procesor normal.
- **PyTorch**: un alt framework major de Deep Learning, dezvoltat de Meta AI Research, cunoscut pentru flexibilitatea sa și abordarea dinamică a grafurilor computaționale
- **scikit-learn**: biblioteca fundamentală pentru Machine Learning tradițional în Python, oferind implementări eficiente ale algoritmilor clasici precum Random Forest, Support Vector Machines, Logistic Regression și k-Nearest Neighbors. În cadrul platformei, scikit-learn joacă un rol crucial prin calculul metricilor de evaluare prin funcții specializate precum confusion_matrix, classification_report, accuracy_score, f1_score.
- **NumPy**: constituie fundația numerică a întregului ecosistem Python pentru calcul științific. NumPy oferă suport pentru array-uri multidimensionale și o colecție vastă de funcții matematice pentru operații vectorizate.

2.4.2.2 Utilizarea TensorFlow in cadrul lucrarii

In implementarea lucrarii, TensorFlow a fost folosit in urmatoarele scopuri:

- Definirea arhitecturilor de rețele neuronale prin API-ul *Keras*, care oferă o interfață de nivel înalt pentru construirea straturilor (*layers*) și modelelor
- Antrenarea modelelor pe datele locale ale fiecărui client FL prin utilizarea de optimizori, precum *Adam*. Un optimizor este o metoda matematica de reducere a ratei de eroare si de imbunatatire a acuratetii parametrilor modelului de invatare automata.
- Extragerea și setarea ponderilor (*weights*) modelului pentru procesul de agregare, operatiune esentiala in FL
- Evaluarea performanței modelului prin metrici precum *accuracy*, *loss*, *precision* și *recall*.

2.4.2.3 Metrici de Evaluare in Machine Learning

Acuratețea (*accuracy*) este definită ca raportul dintre numărul de predicții corecte (atât *true positives*, cât și *true negatives*) și numărul total de predicții:

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{TP + TN}{TP + TN + FP + FN}$$

Recall, sau true positive rate (TPR) este metrica ce determina procentul de date care a fost clasificat corect. Aceasta valoare include la numitor atat numarul de elemente a caror predictie a fost identificata corect drept adevarata si numarul elementelor prezise corect drept incorecte.

$$\text{TPR} = \frac{TP}{TP + FN}$$

Precizia exprimă proporția predicțiilor pozitive ale modelului care sunt corecte:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Anaconda facilitează instalarea și actualizarea rapidă a bibliotecilor necesare, asigurând compatibilitatea între diferite versiuni și reducând riscul apariției conflictelor de dependințe. La nivelul lucrarii, au fost create două medii distincte în Anaconda, fiecare configurat cu bibliotecile relevante pentru proiect, TensorFlow si PyTorch, utilizate in procesul de simulare.

Dincolo de capacitățile sale în Machine Learning, Python excelează și în dezvoltarea de aplicații web și servicii backend. În cadrul platformei, Python îndeplinește multiple roluri care vor fi amintite pe tot parcursul capitolului 3 de implementare.

2.4.3 Rest API

REST (Representational State Transfer) API reprezintă o arhitectură software pentru comunicarea între componente distribuite prin protocolul HTTP. Un API (Application Programming Interface) reprezintă o interfață definita special prin care diferite componente software comunică între ele.

În platforma dezvoltată, REST API asigură comunicația între frontend (React), backend (FastAPI) și orchestrator (Flask), implementând un sistem complet de management al simulărilor federate.

2.4.3.1 Arhitectura REST in Platforma

Platforma utilizează două servere REST API distincte, fiecare cu responsabilități specifice:

Backend API (FastAPI) rulează în containerul Docker local și servește ca intermediar între interfața utilizator și serverul de simulare. FastAPI este un framework Python modern, de înaltă performanță, bazat pe Pydantic pentru validarea automată a datelor și pe specificația OpenAPI pentru documentare automată. Alegerea FastAPI față de Flask pentru backend este justificată de performanța superioară în scenarii cu cereri concurente, suportul nativ pentru operații asincrone.

Orchestrator API (Flask) rulează pe serverul ATM și gestionează execuția efectivă a simulărilor. Flask este un framework web minimalist, ideal pentru servicii interne care nu necesită complexitatea FastAPI. Orchestratorul primește comenzi de la backend, alocă resurse GPU, lansează procese de simulare, și returnează rezultatele.

Un aspect fundamental al arhitecturii este comunicarea asincronă între componente. Simulările FL pot dura de la minute la ore, făcând imposibilă așteptarea sincronă a rezultatelor. Platforma implementează pattern-ul Job Queue cu următoarele caracteristici:

- Inițierea simulării se realizează prin cererea POST către backend la endpoint-ul dedicat, stochează configurația în baza de date PostgreSQL, și transmite cererea către orchestrator.
- Monitorizarea progresului se realizează prin două mecanisme complementare: WebSocket pentru actualizări în timp real și polling HTTP pentru backwards compatibility.

Backend-ul expune multiple categorii de endpoint-uri, fiecare responsabil de o funcționalitate specifică:

- Autentificare și Autorizare se realizează prin endpoint-uri dedicate pentru înregistrare utilizatori, autentificare cu username și parolă, și generare JWT (JSON Web Token) pentru sesiuni. Token-ul JWT este inclus în header-ul Authorization pentru toate cererile ulterioare, sub forma Bearer token. Backend-ul validează token-ul pentru fiecare cerere protejată, verificând semnătura digitală, expirarea, și existența utilizatorului în baza de date.
- Management Proiecte și Fișiere permite utilizatorilor să organizeze simulările în proiecte ierarhice. Fiecare proiect poate conține multiple fișiere reprezentând template-uri Python pentru modele diferite.
- Execuția Simulărilor este orchestrată prin endpoint-ul principal care primește codul template Python și configurația simulării. Backend-ul efectuează validări de securitate asupra codului încărcat pentru a preveni atacurile de tip code injection, verifică existența funcțiilor obligatorii în template și transmite cererea către orchestrator împreună cu token-ul de autentificare.

- Colectarea Rezultatelor se realizează prin endpoint-uri care interogează orchestratorul pentru obținerea rezultatelor finale. După ce orchestratorul semnalează completarea simulării, backend-ul solicită fișierele JSON cu metrici, fișierul text cu rezumatul, și informații despre atacul aplicat.
- WebSocket pentru Monitorizare Real-Time implementează un endpoint special care acceptă conexiuni WebSocket. După stabilirea conexiunii, backend-ul intră într-un loop asincron în care interoghează orchestratorul la fiecare 2 secunde pentru status-ul actualizat al simulării. Informațiile sunt trimise imediat către frontend prin WebSocket, oferind o experiență interactivă similară unui terminal live.

Capitolul 3:

Proiectare, Implementare si Testare

Lucrarea se concentreaza pe dezvoltarea unei platforme de simulare a atacurilor de tip Data Poisoning intr-un mediu de invatare automata federata. Scopul ei vine chiar din problema identificarii acestui tip de atac si imbunatatirea procesului de alertare in securitatea cibernetica. Platforma sprijina analistii de securitate si cercetatorii de analiza de date in vederea constientizarii unei posibile vulnerabilitati la nivelul unei infrastructuri publice centralizate.

3.1 Arhitectura Platformei

In procesul de dezvoltare al platformei, au fost analizate platforme de programare in timp real, precum Google Colab sau Jupyter Notebook, care permit utilizatorilor sa isi incarce si sa execute remote codul de antrenare a retelelor neuronale intr-o infrastructura specializata. Inspirandu-se din aceste platforme si observand existenta unor utilitare de analiza a malware-ului si infectiilor, s-a identificat nevoia unui framework care nu doar sa simuleze o retea de invatare automata federata, ci si sa testeze atacuri Data Poisoning oferind cercetatorilor si analistilor de securitate posibilitatea de a evalua vulnerabilitatile propriilor retele neuronale.

Platforma este structurată pe trei niveluri principale:

Nivelul Client (Frontend)

- Interfață React pentru autentificare, management proiecte, și vizualizare rezultate
- Comunicare prin REST API și WebSocket pentru monitoring real-time

Nivelul Backend (Containerizat Docker)

- API FastAPI pentru gestionarea cererilor utilizatorilor
- Baza de date PostgreSQL pentru persistența configurațiilor și rezultatelor
- Validare securitate cod și transmitere asincronă către orchestrator

Nivelul Execuție (Server ATM)

- Orchestrator Flask pentru managementul simulărilor
- Alocare dinamică GPU prin GPUManager (3 GPU-uri disponibile)
- Medii Python izolate: fl_tensorflow și fl_pytorch

Fluxul platformei este urmatorul:

- Utilizatorul se autentifica in platforma si isi creeaza un proiect si un fisier de test aferent, in care incarca un sablon specific pentru antrenare. Sablonul este compus dintr-o serie de functii predefinite care vor trebui completate cu secvente de cod specifice crearii retelei neuronale, antrenarii si evaluarii ei. Printre alte functii, trebuie sa amintim si de cele de setare a ponderilor pentru procesul de agregare la nivelul infrastructurii federate, de descarcare de date local de la o sursa specifica si de salvare a configuratiei retelei antrenate.
- Odata ce se initiaza simularea la apasarea butonului Run, sablonul python este transmis asincron de la backend-ul aplicatiei catre un server remote. Comunicarea asincronă permite utilizatorului să continue lucrul în platformă în timp ce simularea se execută pe server. Serverul remote este denumit Server ATM, fiind localizat in infrastructura Academiei Tehnice Militare "Ferdinand I". Pe serverul remote, cererea este preluata de un proces orchestrator care odata primita, instantiaza un nou proces care executa o serie de actiuni.

- Odata initiata o cerere de simulare, se va crea in directorul utilizatorului logat in platforma, un director specific denumit dupa identificatorul taskului instantiat de utilizator. In acel director, se afla toate tipurile de fisiere care vor fi gestionate pe tot parcursul operatiunile de simulare, de la setul de date de antrenare, pana la fisierele specifice tipului de atac Data Poisoning utilizat si rezultatele simularilor in reteaua federata.

- Prima etapa este cea in care se identifica tipul de biblioteci utilizate pentru procesarea retelei neuronale si se trece in mediul python corespunzator. Platforma suporta antrenarea folosind biblioteci TensorFlow si PyTorch, avand create cate un mediu specific pentru antrenarea tipului de retea respectiv. Scriptul de rulare a retelei neuronale se numeste train_model.py si este un script agnostic, construit pentru a testa functiile din sablon.

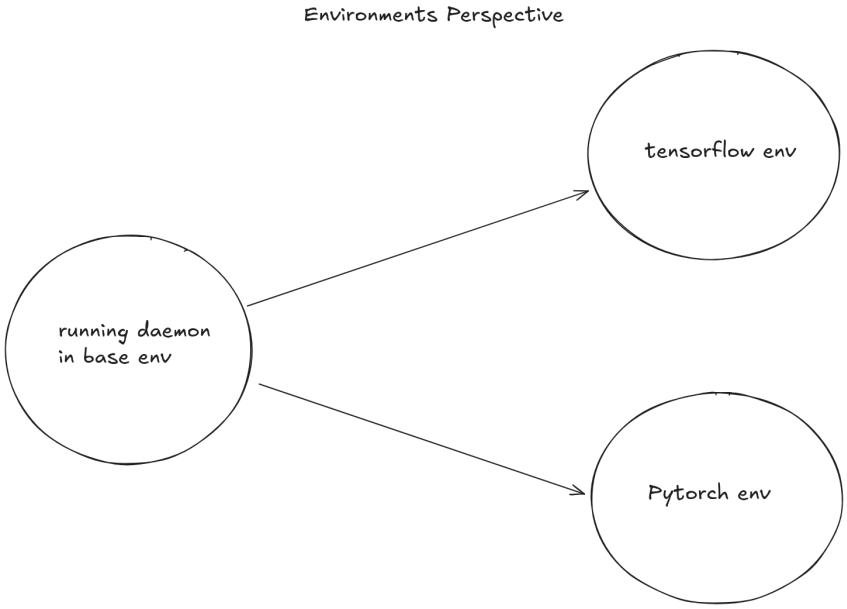

Figura 3.1: Medii de python platforma

- Imaginea de mai sus reprezinta mediile de python disponibile pentru a rula retelele neuronale.

- La fiecare pas, in cazul unei erori generate de orice fel de actiune, procesul ce gestioneaza operatiunile de simulare se opreste si se transmite eroarea ca rezultat procesului orchestrator care semnalizeaza backend-ul aplicatiei.

- Dupa prima antrenare, se salveaza parametrii aferenti matricii de confuzie (acuratete, precizie, scor F1, recall) pentru a compararea rezultatului simularii cu configuratia initiala a retelei si detectarea posibilelor anomalii. Acesta este principalul pas in procesul de analiza a problemei propuse.

- Daca programul este corect structurat, procesul curent incarca functiile scriptului transmis si se foloseste de ele pentru a simula atacul Data Poisoning. Mai intai se apeleaza functia de descarcare locala a datelor de antrenare si testare. Aceste date vor fi folosite pentru a simula procesul de otravire pe baza configuratiei simularii setate de utilizator. Datele pot fi descarcate din orice sursa publica, permitand o flexibilitate in tipul de platforma si de date pentru testare.

- Argumentele cu ajutorul carora se ruleaza scriptul de otravire a setului de date sunt:

  - **poison_operation**: tipul atacului (noise, label_flip, backdoor)
  - **poison_intensity**: intensitatea atacului (ex: 0.1 pentru zgomot Gaussian)
  - **poison_percentage**: procentajul datelor afectate (ex: 0.2 pentru 20%)

- Amintim ca mediile de python includ si biblioteci de seturi de date specifice precum tensorflowDatasets in care se regasesc seturi de date predefinite in cazul in care utilizatorul doreste sa foloseasca un set de date predefinit la nivel de biblioteci.

- Procesul de otravire (poisoning) presupune rularea unui script care pe baza parametrilor simularii, aplica o serie de operatii asupra seturilor de date. Tipurile de actiuni de otravire au fost definite in capitolul 2 (sectiunea 2.3.3 - Atacul Data poisoning). Utilizatorul poate alege procentajul din etichete care sa fie infectate, iar in cazul zgomotului Gaussian si a backdoor, se va seta si intensitatea pixelilor de alterat.
- In pasii urmatori, se va realiza simularea propriu-zisa cu ajutorul unui script agnostic denumit fl_simulator, care va primi drept argumente urmatoarele:
  - N numarul de clienti/dispozitive aferente retelei federate
  - M numarul clienti compromisi din totalul de mai sus
  - R numarul total de runde de antrenare
  - ROUNDS numarul de runde in care dispozitivele compromise vor antrena cu datele corupte
  - strategia de propagare a dispozitivelor compromise:
    - *first*: dispozitivele compromise vor rula primele ROUNDS runde cu datele otravite, iar restul de R-ROUNDS runde vor folosi datele curate
    - *last*: dispozitivele compromise vor rula ultimele ROUNDS runde cu datele otravite, iar restul de R-ROUNDS runde vor folosi datele curate
    - *alternate*: dispozitivele compromise vor rula cate o runda folosind datele otravite si cate una cu datele curate
- Prima rulare va fi de fapt o reglare fina (fine-tuning) a configuratiei retelei initiale, folosind drept date de antrenare datele curate. Este important de menționat că ambele simulări (clean și poisoned) pornesc de la **aceeași configurație inițială** a modelului. Acest aspect asigură că diferențele observate în rezultate se datorează exclusiv atacului Data Poisoning, nu variațiilor aleatorii în inițializarea ponderilor. Ambele simulări folosesc procesul de agregare FedAvg pentru a combina update-urile de la clienți la fiecare rundă.
- Urmatorul pas este reglarea fina a aceleiasi configuratii initiale, dar folosind datele otravite in procesul de antrenare.
- In urma acestor simulari, se vor salva informatiile cu privire la fiecare configuratie simulata, ce va fi utilizata in procesul de analiza.

  Analiza rezultatelor presupune compararea următoarelor metrici între simularea cu date curate și simularea cu date otravite:
  - **Degradarea accuracy-ului**:
    - Degradare absolută: $\Delta_{acc}^{abs} = Acc_{clean} - Acc_{poisoned}$
    - Degradare procentuală: $\Delta_{acc}^{\%} = \frac{Acc_{clean} - Acc_{poisoned}}{Acc_{clean}} \times 100\%$
  - **Metrici de clasificare**: Precision, recall și F1-score pentru evaluarea impactului atacului
  - **Convergență**: Deviația standard și tendința accuracy-ului în ultimele 5 runde
  - **Divergența ponderilor**: Distanța euclidiana medie între actualizările clienților la fiecare rundă
- Utilizatorul va putea vizualiza fiecare pas de procesare amintit pana acum si rezultatul analizei simularii.
- Pentru a facilita munca analistului, platforma include posibilitatea de export a raportului simularii, dar si compararea de simulari ale aceluiasi proiect.

3.2 Implementare Platformei

Platforma este compusă din două servere, unul dintre ele fiind remote și dispunând de trei plăci grafice NVIDIA A40 (GPU) optimizate pentru medii de lucru cu Inteligență Artificială și Deep Learning. Plăcile dispun de 46 GB memorie VRAM (video RAM) GDDR6 pentru a suporta lucrul intensiv cu date. A40 este o placă profesională destinată centrelor de date, furnizând accelerare hardware pentru TensorFlow

și PyTorch, precum și pentru operații matematice complexe necesare în procesele de simulare și învățare federată.

Serverul care rulează funcționalitățile backend și de persistență se află în rețeaua locală proprie. Acesta este responsabil pentru gestionarea interacțiunii utilizatorilor, stocarea configurațiilor și transmiterea cererilor către serverul remote. El reprezintă punctul de intrare al platformei pentru toți utilizatorii și menține logica de securitate, autentificare și gestionare a sesiunilor.

Pentru scalabilitate și flexibilitate, s-au utilizat containere Docker la nivelul serverului local pentru a stoca următoarele componente:

- **Containerul frontend** - include toate dependențele necesare pentru rularea unei aplicații React folosind un server NodeJs. Acesta oferă o interfață modernă și receptivă, permițând utilizatorilor accesarea tuturor funcțiilor platformei, precum încărcarea de șabloane, vizualizarea rezultatelor și compararea simulărilor.
- **Containerul bazei de date PostgreSQL** - stochează toate fișierele cu rezultatele simulărilor executate pe server, specifice fiecărui utilizator, precum și parolele hash ale utilizatorilor. Structura bazei este optimizată pentru volum mare de date și interogări rapide asupra istoricului de simulări.
- **Containerul backend** - procesul responsabil de primirea cererilor API. Backend-ul recepționează comenzi de la utilizatori sub forma unei cereri POST către endpoint-ul `/run`. După validare, cererea este retransmisă asincron către serverul remote. Backend-ul implementează validări de securitate asupra șabloanelor, logarea fiecărei cereri și mecanisme de throttling pentru prevenirea supraîncărcării serverului remote.

La nivelul serverului remote, platforma include următoarele componente:

- **Procesul orchestrator** - rulează scriptul `orchestrator_gpu.py`, care creează un proces nou pentru fiecare conexiune inițiată. Pentru fiecare simulare se alocă un GPU, asigurând un flux de procesare ridicat fără a afecta timpul de execuție al simulărilor deja active. Orchestratorul gestionează coada de task-uri, starea proceselor și comunicarea bidirecțională cu serverul local.
- **Managerul de GPU** - implementat în scriptul `gpu_manager.py`, monitorizează în timp real starea GPU-urilor (nivel de utilizare, memorie liberă, temperatură) și distribuie simulările în mod echilibrat pentru a preveni supraîncărcarea unuia dintre dispozitive.
- **Procesul de simulare a atacului** - identifică tipul bibliotecilor Python folosite în șablonul primit și comută automat în mediul Anaconda corespunzător (TensorFlow sau PyTorch). Mediile Python sunt complet izolate, având propriile dependințe pentru a asigura reproducibilitatea rezultatelor. Otrăvirea datelor este realizată prin scriptul `poison_data.py`. Simularea propriu-zisă rulează prin `fd_simulator.py`, care utilizează șablonul de cod stocat în `template_code.py`. În timpul execuției, fiecare etapă este jurnalizată, iar eventualele erori sunt transmise backend-ului în timp real.

Arhitectura propusă oferă următoarele beneficii:

- **Flexibilitate**: Sistemul de șabloane permite testarea oricărui model TensorFlow sau PyTorch fără modificări ale infrastructurii. Izolarea mediilor Python elimină conflictele dintre versiuni diferite ale bibliotecilor.
- **Scalabilitate**: Alocarea dinamică a GPU-urilor permite rularea simultană a trei simulări, maximizând utilizarea resurselor disponibile. Structura modulară permite extinderea ulterioară cu mai multe GPU-uri sau chiar cu mai multe servere.
- **Reproducibilitate**: Toate configurațiile și rezultatele sunt stocate în PostgreSQL, permițând replicarea exactă a experimentelor. Mediile conda imutabile garantează consistența dintre execuții.
- **Transparență**: Monitorizarea în timp real prin WebSocket oferă vizibilitate completă asupra procesului de execuție, facilitând depanarea eventualelor probleme.
- **Comparabilitate**: Funcționalitatea de comparare a simulărilor permite evaluarea impactului diferitelor tipuri de atacuri asupra aceluiași model, datele fiind centralizate într-o structură unificată pentru analize statistice avansate.

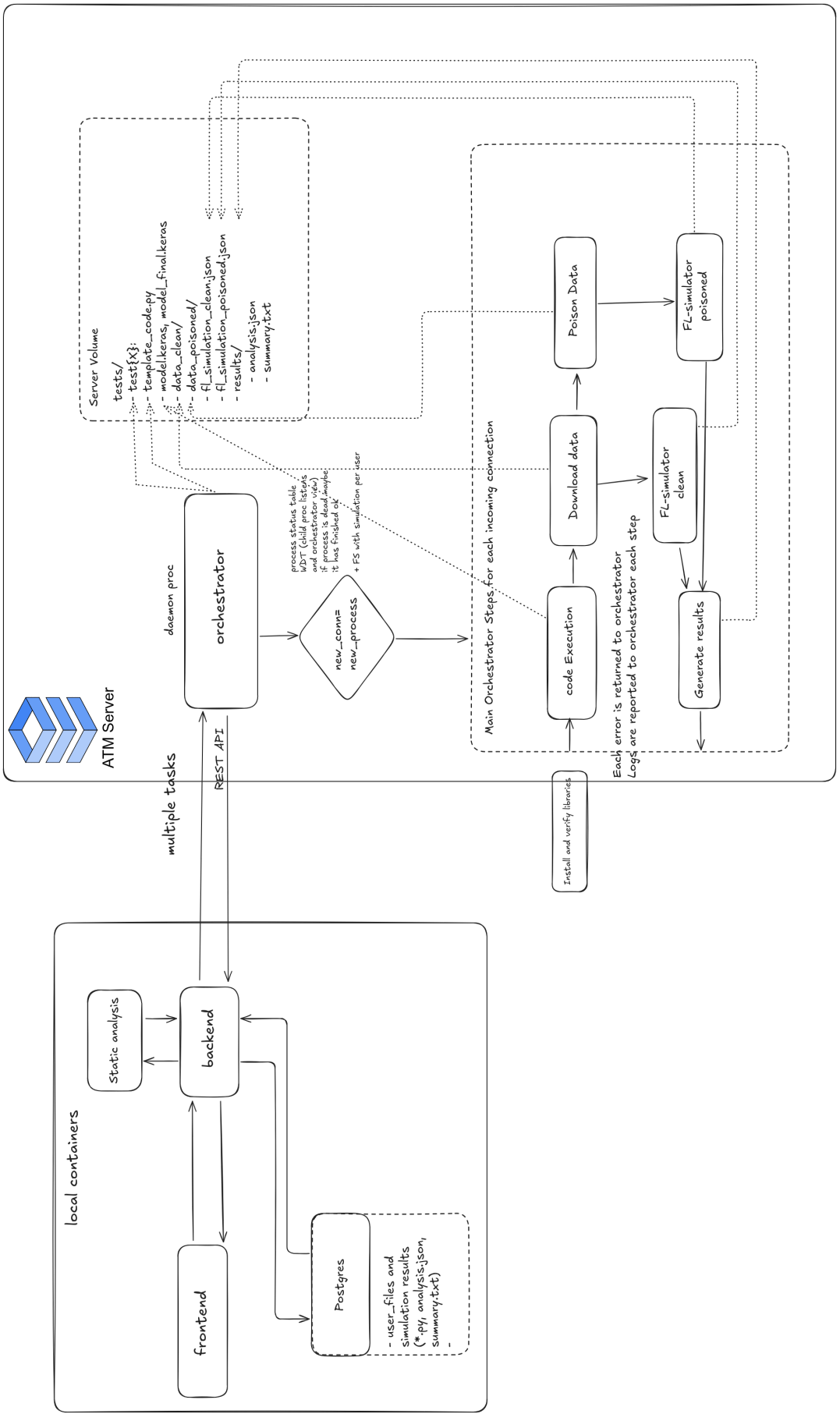

Figura 3.2: Arhitectura si fluxul platformei

Imaginea 3.2 prezinta intregul flux al arhitecturii, precum si detaliile tehnice utilizate.

3.3 Cerintele Software

3.3.1 Cerintele functionale

3.3.2 Cerintele nefunctionale

3.4 Arhitectura platformei

3.4.1 Containere

3.4.2 Server

3.5 Testare

Capitolul 4:

Rezultate si Metrici Simulari

4.1 Evaluare Performante

4.1.1 Scalabilitatea Simularilor

4.1.2 Scalabilitatea platformei

4.2 Evaluare Rezultate

4.2.1 Performante Gaussian Noise

4.2.2 Performante Label-Flip

4.2.3 Performante Backdoor

Capitolul 5:

Concluzii si dezvoltare ulterioara

5.1 Starea Curenta

5.2 Dezvoltare Ulterioara

5.3 Tabele

Tabelele sunt aranjări a informației într-o structură formată din linii și coloane, care permite o mai bună observare a acesteia.

Mai jos apar două exemple. Primul tabel este de dimensiune mică. Al doilea, din cauza dimensiunii mai mari, are o orientare inversată și este plasat singur pe o pagină.

Nume Complet	Funcție Ocupată
Joshua Roob	Manager de Proiect
Asa Hauck	Artist Grafic
Harley Hagenes	Programator

Tabel 5.1: Colaboratori la Realizarea Studiului

Stat	Oraș	Latitudine	Longitudine
South Carolina	Corwinberg	86.609523	42.408007
Rhode Island	East Isaacmouth	63.17309	-13.786023
Mississippi	North Noblestad	-31.316834	5.280483
Illinois	Grahamland	-39.853659	-77.713676
Rhode Island	West Richardfurt	67.583131	31.858455
Florida	Port Roberta	25.276026	83.715344

Tabel 5.2: Locații de Conducere a Studiului

NECLASIFICAT

5.4 Imagini

Imaginile sunt utilizate în cadrul lucrării pentru exemplificarea unor idei în manieră vizuală.

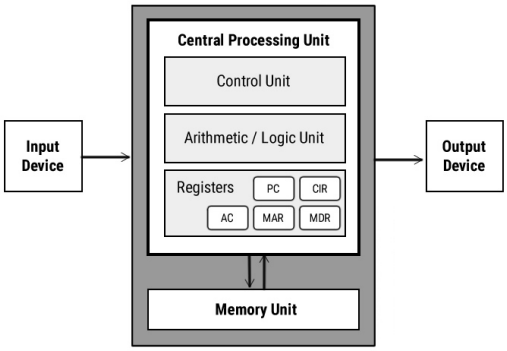

Figura 5.1: Arhitectura unui calculator[1]

5.5 Liste

Listele sunt simple serii de informații.

- Un item
- Unul dintre itemi
- Încă un item

Acestea pot conține itemi identificați prin numere dacă indexarea sau sortarea sunt necesare.

1. Primul item
2. Al doilea item
3. Al treilea item

5.6 Formule Matematice

LaTeX oferă un mod programatic de a construi formule matematice, după cum este cea de mai jos.

[1] Arhitectura ilustrată este de fapt cea von Neumann.

$\sum \mathbf{F} = 0 \Leftrightarrow \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = 0$

5.7 Note de Subsol. Citări

Notele de subsol pot fi utile în cazul explicațiilor suplimentare (cum a fost cea referitoare la imaginea inclusă, la care sintaxa este puțin diferită din cauza plasării notei în cadrul legendei) sau a citărilor[2] care nu se pretează a fi trecute în bibliografie din cauza utilizării lor punctuale.

Pe de altă parte, sursele bibliografice citate intens [1] sunt marcate corespunzător și notate în bibliografie.

5.8 Etichete. Referințe

În cadrul surselor LaTeX a acestui document, apar *tag*-uri `\label` care creează o etichetă utilă referințelor interne. Acestea din urmă indică elemente din cadrul documentului curent (de exemplu, către tabelul 5.1).

Mai pot apărea referințe externe, către resurse din Internet (de exemplu, către *website*-ul Wikipedia).

[2] Cristian Lupascu, Cezar Plesca și Mihai Togan, "Privacy Preserving Morphological Operations for Digital Images", în (iun. 2020), pp. 183–188, DOI: 10.1109/COMM48946.2020.9141997

Bibliografie

Cărți

[1] Mihai Togan, *Cryptographic Technologies for Data Protection in Cloud*, Editura Matrix Rom, 2017.

Articole Științifice

[2] Ionuț Dumitru și Mihai Togan, "Client Module with Multifactor Authentication for Remote Electronic Signature Generation Using Cryptography API: Next Generation", în *Journal of Military Technology* 3 (iun. 2020), pp. 5–10, DOI: 10.32754/JMT.2020.1.01.

[10] Peter Kairouz et al., "Advances and Open Problems in Federated Learning", în *Foundations and Trends® in Machine Learning* 14.1-2 (2021), pp. 1–210, DOI: 10.1561/2200000083, URL: https://doi.org/10.1561/2200000083.