Doktori (PhD) értekezés tézisei

Gépi tanulást használó alkalmazások tervezése és teljesítményelemzése

Szabó Máté

Témavezető: Dr. Ispány Márton

DEBRECENI EGYETEM

Informatikai Tudományok Doktori Iskola

Debrecen, 2025

Tartalom

Bevezetés és motiváció

Háttér

Az adat a tudomány és a mindennapok alapja, amely a technológia fejlődésével a 21. század kulcsfontosságú erőforrásává vált. A gépi tanulás lehetővé teszi ezen adatok értelmezését, három fő formában: felügyelt, felügyelet nélküli és félig felügyelt tanulás. A szoftverfejlesztésben a web- és mobilalkalmazások terjedése platformfüggetlen, könnyen frissíthető és személyre szabható megoldásokat kínál. A dolgozat központi témája a gépi tanulás, valamint webes és mobilalkalmazások integrációja. Bemutatott esettanulmányai azt szemléltetik, miként építhetők intelligens, skálázható és felhasználóbarát rendszerek, ahol a gépi tanulás bármely internetkapcsolattal rendelkező szoftverbe beépíthető.

Gépi tanulás okostelefonokon: nagy adathalmazok feldolgozása elosztott architektúrában

A gépi tanulás mára szinte minden digitális szolgáltatásban megjelenik, alkalmazási területei az egészségügytől az önvezető autókon át a kereskedelemig terjednek. Az orvosi példák közé tartozik a rák előrejelzése [1], szívbetegségek diagnosztikája [2], az Alzheimer-kórhoz köthető változások vizsgálata [3], valamint betegségek korai felismerése [4]. További alkalmazások az arcfelismerés [5] és az internetes forgalom osztályozása [6]. A növekvő adatmennyiség új, hatékonyabb algoritmusokat igényel. A hagyományos módszerek – döntési fák [7], Bayes-hálók [8], SVM [9] – bár skálázhatók, korlátozottak nagy adatállományok esetén.

A mobiltelefonok mára a legelterjedtebb számítástechnikai eszközökké váltak, folyamatosan növekvő számítási kapacitással [10, 11]. Sok készülék már képes ML-modellek futtatására, miközben erőforrásaik gyakran kihasználatlanok maradnak [12–14].

Az általam kidolgozott prototípus két fő részből áll:

- **Webszolgáltatás**: modellek fogadása, mentése, ensemble modellek képzése (amelyek javíthatják a pontosságot [15]), valamint adatfeldolgozás.
- **Mobilalkalmazás**: egyszerű neurális hálók tanítása, előtanított modellek használata, kommunikáció a szerverrel.

A munka kapcsolódik a mobil és elosztott gépi tanuláshoz, valamint a modellcsere-formátumokhoz:

- **Modellcsere**: például PMML [16] és ONNX [17, 18].
- **Mobil ML**: képfelismerés, rosszindulatú szoftverek detektálása [19–21].
- **Webszolgáltatáson keresztüli ML**: előtanított modellek elérése korlátozott eszközökről [22].
- **Elosztott ML**: paraméter szerverek [23], valamint MLBase [24], MLI API [25], Spark MLlib [26].

Gépi tanulás mobil eszközökön és Java alapú megoldások

A mobil eszközök fejlődésével egyre fontosabbá vált a gépi tanulás (ML) szerepe a személyre szabott felhasználói élményben. A legtöbb alkalmazás előre betanított modelleket használ (pl. hangfelismerés, képfeldolgozás), de a mobilon történő tanítás energiaigényes és kihívásokkal teli [27]. Ennek ellenére hasznos felhasználások léteznek, mint a mobilprocesszorok teljesítményének összehasonlítása [28], úthibák felismerése [29] vagy rosszindulatú szoftverek detektálása [21].

A disszertáció 3. fejezete a gépi tanulás és az Android benchmarkolás kapcsolatát vizsgálja. A szakirodalomban több benchmark rendszer található, például PMLB [30], DAWNBench [31], MLPerf [32] vagy Sentinel-2 képosztályozás [33].

Bár a Java nem domináns nyelv az ML-ben, alkalmazásfejlesztésben továbbra is meghatározó. Számos Java-alapú könyvtár érhető el:

- **Weka** [34]: teljes ML-szoftver grafikus és könyvtár formában.
- **SMILE** [35]: nagy teljesítményű motor, széles algoritmusválasztékkal.
- **Oracle Tribuo** [36]: provenance, típusbiztonság és interoperabilitás; támogatja XGBoost [37], TensorFlow [38] és ONNX [18] integrációját.
- **Deeplearning4J** [39]: mélytanulásra fókuszáló könyvtár.
- **H2O** [40]: memóriaalapú platform, REST API integrációval.
- **Mallet** [41]: NLP-re és dokumentumosztályozásra specializált.

Mikroszolgáltatások és gépi tanulás integrációja

Számos kutatás foglalkozott a gépi tanulás és mikroszolgáltatások kapcsolatával. Egyesek az ML-t szolgáltatásként definiálják REST-interfészen keresztül [42], mások mobil eszközökre adaptálják (pl. DroidAutoML, Android kártevők felismerése [43]). Vannak általános ML-folyamatokat támogató architektúrák [44], illetve ensemble megoldások, ahol több modell és API-gateway működik együtt [45]. A cCube architektúra [46] központi orchestrator szolgáltatással vezérelt rendszert kínál, hasonlóan kutatásunkhoz.

Az egészségügyben a KETOS [47] döntéstámogató rendszer, míg a szoftverfejlesztésben ML segíthet mikroszolgáltatások osztályozásában és erőforráskezelésben [48–50]. Az architektúra-tervezést frontend-irányelvek [51] és minták (API gateway, aggregator [54, 55]) segítik.

A mikroszolgáltatások előnyeik mellett komplexitást, verziókezelési és monitorozási nehézségeket is hordoznak [52]. Az elosztott ML ráadásul nagyobb számítási időt, energiaigényt és bonyolultabb párhuzamos kezelést eredményezhet [53].

Gépi tanulást megvalósító szoftverkomponensek integrációjára két modellt azonosítottunk:

- **Gateway modell**: aggregáló szolgáltatás fogja össze az eredményeket.
- **Direkt modell**: a kliens kapja a rész-eredményeket és maga egyesíti azokat.

Mivel az ilyen szoftverkomponensek közös adatbázisa sértené a microszolgáltatás architektúra egyik előnyeként definiált függetlenséget, minden szolgáltatás saját adatbázist használ.

Referenciaimplementációnk Spring Boot és Spring Cloud alapú, Eureka Serverrel a szolgáltatás-felfedezéshez [56]. A Java-szolgáltatások Weka [34] és Deeplearning4j [38] modelleket, a Python-szolgáltatások (Scikit-learn [57], TensorFlow [39], Azure ML [58]) Flask keretrendszert alkalmaznak.

A kutatásunk

A kutatási munkát 2018-ban kezdtem meg, egyszerre két szálon, mivel úgy éreztem, ezek egy ponton majd a tanulmányaim végén össze fognak érni. Az egyik célja a gépi tanulási algoritmusokkal való kísérletezés különböző alkalmazásokon és adatokon, a másik pedig egy olyan általánosabb keretrendszer vagy referencia implementáció, amely lehetővé teszi egyszerűbb eszközök számítási kapacitásait összehangolni modelltanítás céljából, mindezt lehetőleg platformfüggetlenül.

Az első kísérletező projekt az SMNIST volt, ahol a számosság felismerését hasonlítottuk össze emberi és gépi környezetben. Mivel itt többen vizsgáltuk ugyanazokat a különböző nehézségű adathalmazokat, lehetőségem volt megismerni több platformot, eszközt, amellyel javarészt konvolúciós neurális hálókat lehet tanítani. Itt már felmerült az a probléma, hogy mivel mindenki saját környezetben tanította a

modelljét különböző nyelveken, ezeket nem lehet könnyen egy másik feladatra implementálni, tehát nincs egységeges elérhetőség.

Az egységesebb elérhetőséghez az első állomás a modell és adatpárhuzamosság fogalmainak megismerése volt. Első körben az adatpárhuzamos megoldást valósítottam meg, környezetnek pedig egy félig független megoldást választottam a Java nyelvvel és az Android platformmal. A cél itt az volt, hogy a nagyobb adathalmazt felszeletelve kisebb modelleket hozok létre asztali és mobil környezetben, majd ezeket egy közös helyen kezelem. Erre jelentett megoldást a bytekód szerializáció, ami a betanított modelleket byte tömbbe írta, majd egy szintén Java környezetben összeállította, így egy helyen lehetett a paramétereiket kezelni vagy ensemble modellbe befoglalni.

Ahogy a gépi tanulási platformok is egyre kiterjedtebbek lettek, így újféle, modernebb környezetekben lehetett a meglévő algoritmusokat tesztelni. Ez hozta életre a Red Flower Hell projektet, aminek az egyik célja a gépi tanulási eszközök iránti igényt kialakítani az oktatás során. A környezetet itt a Microsoft Minecraft Malmö adta, aminek a lényege, hogy a Minecraft játékon belül egy speciális feladat megoldását kell leprogramozni egy ágensnek. Ez egy remek gyakorlóterep volt a magas szintű programozást tanuló hallgatóknak, akik ezen eszköz segítségével Pythonban egyszerű algoritmusok segítségével próbáltak versenyezni egymással. Mivel ez a feladat hagyományos algoritmusokkal nem oldható meg hatékonyan, az AI eszközökkel való ismerkedés elengedhetetlenné vált.

A mobil platformos adatpárhuzamos megoldás után szerettem volna, ha a keretrendszerem alapértelmezés szerint a mobilon leghatékonyabb gépi tanulási könyvtárat támogatja, így 4 feltétel mentén megvizsgáltam a 3 legnépszerűbbet: az Oracle Tribuo-t, a Weka-t, és a SMILE-t. A kísérlethez különböző méretű és komplexitású adatállományokat választottam ki, majd ezekre vizsgáltam a 3 könyvtár azonos

algoritmusait. Az eredmények vegyesek voltak, mint a futásidő, memóriaigény, akkumulátorigény, átlagos cpu használat tekintetében, de összességében a korábbi munkában is használt WEKA teljesített a legjobban.

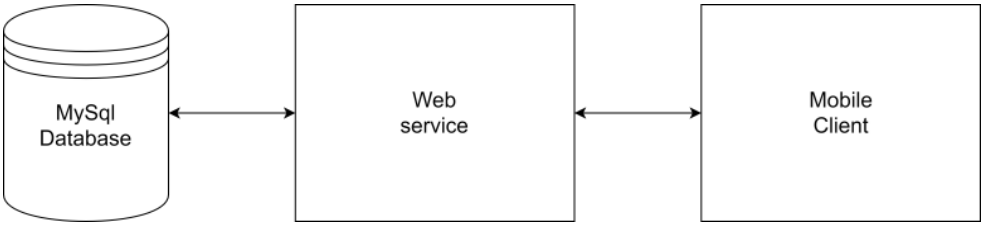
Mivel a korábbi munkák erősen Java alapúak voltak, ezért a platform és nyelvfüggetlenség felé kellett tovább haladni, ehhez viszont el kellett engedni azt a koncepciót, hogy az összes modellt egy közös rendszerben kell kezelni. Az új irány egy lazábban csatolt, könnyebben bővíthető megoldás, amit a mikroszolgáltatások inspiráltak. Mivel ezek természetükből adódóan különböző platformokon, nyelveken, külön adatforrással íródnak, ezért a modellek közös formátumra való konvertálását el lehetett hagyni. A koncepció itt az, hogy különböző környezetekben elkészülnek a betanított modellek, amelyek mikroszolgáltatásként publikálódnak, így bármilyen más, a rendszeren belüli szoftver tudja őket használni. A több modell összevonása helyett pedig ensemble modellekkel lehet a meglévőket kombinálni. A kérdések itt főképp az adatok, modellek elérhetőségére, illetve a számítási teher viselésére vonatkoznak, ezeknek a finomhangolásával lehet egyedi igényre szabni a rendszert, akár szerveroldali, akár kliensoldali teherrel.

Az értekezés új tudományos eredményei és tézisei

Adatpárhuzamos gépi tanulás mobil eszközökön

1. Tézis: A mobil eszközök elosztott gépi tanulásba való bevonása adat-párhuzamosság révén technikailag megvalósítható. Bár a teljesítmény nagymértékben a felhasznált könyvtárak implementációjától függ, a mérések azt mutatják, hogy kisebb adatrészletek feldolgozása megbízhatóan és hatékonyan elvégezhető mobil platformokon.

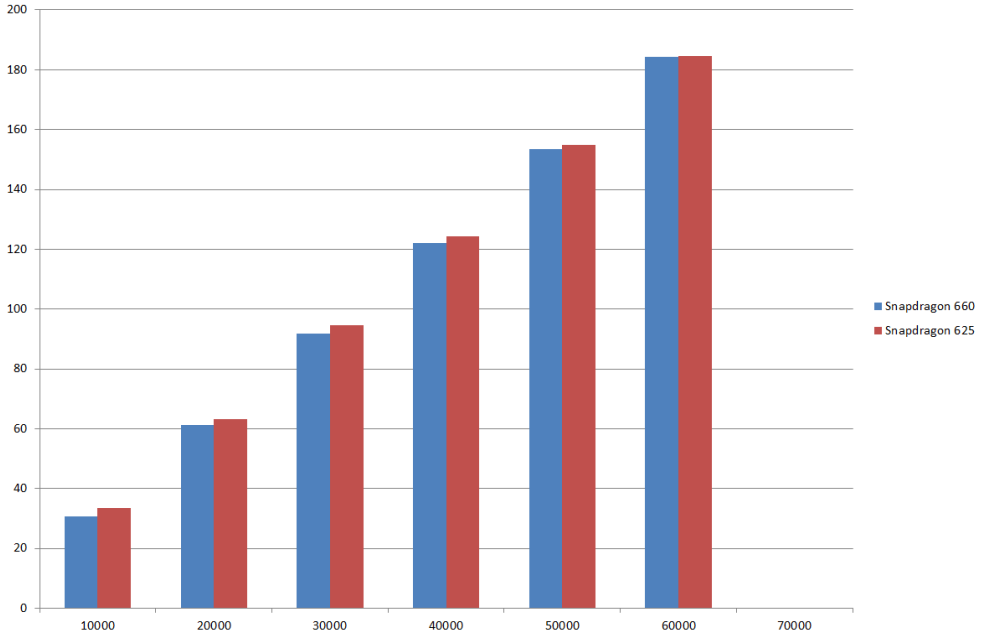
A mobil eszközök elosztott gépi tanulásba való bevonásának lehetősége adat-párhuzamosság segítségével a 2. fejezetben került bemutatásra. Ez működő architektúrát ismertet (1. ábra), amely egy szerveroldali webszolgáltatást, egy relációs adatbázist és Android-alapú klienseket egyesít. Ebben a keretrendszerben a mobil eszközök képesek adathalmaz-részleteket kérni, azokon modelleket tanítani, majd a betanított modelleket visszaküldeni a szerverre aggregáció céljából. Ez az architektúra biztosítja, hogy a mobil eszközök – korlátozott erőforrásaik ellenére – technikailag megbízható módon járuljanak hozzá az elosztott tanítási folyamathoz.

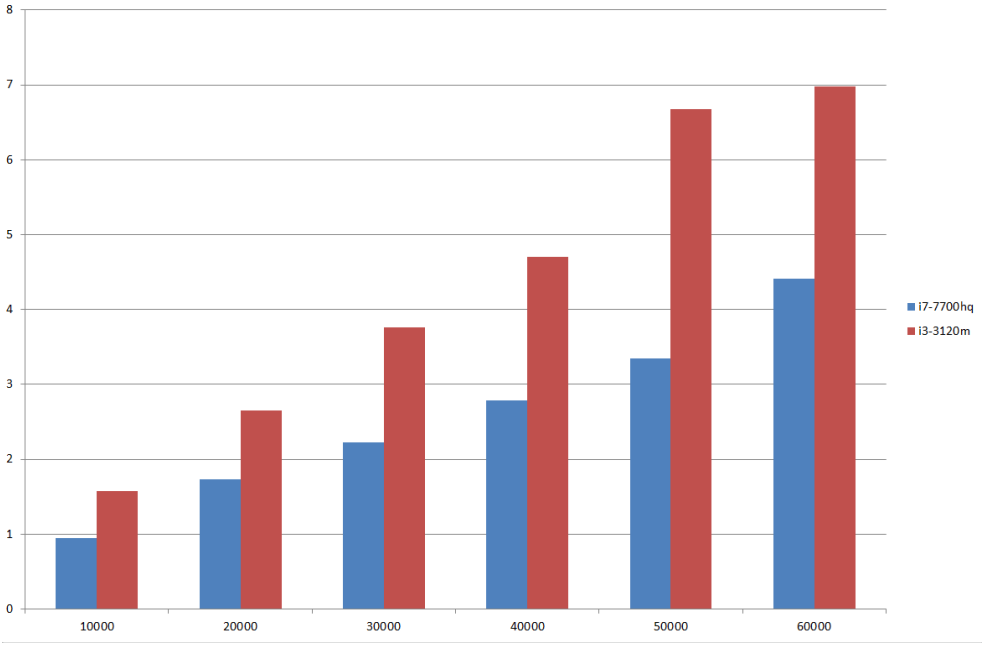


1. ábra - A rendszer komponenseinek kapcsolatai

Az eredmények azt mutatják, hogy a rendszer helyesen működik, és érvényes modelleket állít elő. Az adatfelosztás alkalmazásával még nagyon nagy adathalmazok, például a több millió rekordot tartalmazó

Record Linkage Comparison Patterns dataset is részben feldolgozhatóvá vált mobil eszközökön. Bár az asztali rendszerek természetesen jóval erősebbek, a mérések igazolták, hogy a kisebb adatrészletek – körülbelül 70 000 rekordig – sikeresen betaníthatók okostelefonokon is (2. ábra). A modellek kiértékelése pedig azt mutatta, hogy a mobilon tanított almodellek pontossága, precision és recall mutatói összevethetők a desktop környezetben tanítottakéval (1. táblázat). Ez rámutat arra, hogy a megközelítés nemcsak működőképes, hanem a bizonyos esetekben hatékony is lehet, hiszen a mobil almodellek aggregálhatók nagyobb ensemble modellekké , miközben az előrejelzési teljesítmény akár azonos is maradhat.





2. ábra - Egy epoch futási ideje az adathalmaz méretének függvényében: a) mobilkörnyezetben, b) asztali környezetben

1. táblázat - Pontosság és legjobb eredmény mobil- és asztali környezetben

environment	accuracy	score at the best epoch
mobile	0.7907	4.81924921875
computer	0.7907	4.81931338500

A fejezet ugyanakkor rámutat arra is, hogy a teljesítmény nagymértékben a választott gépi tanulási könyvtáraktól függ. A Deeplearning4J könyvtár került felhasználásra, mivel kompatibilis volt a szerverrel és az Android környezettel, de mobil eszközökön hamar nyilvánvalóvá váltak a korlátai. A többszálúság hiánya és a többmagos architektúrák nem megfelelő kihasználása viszonylag lassú tanítási időt eredményezett. Így a rendszer összteljesítménye kevésbé a telefonok

nyers hardverpotenciáljának, sokkal inkább a használt könyvtárak hatékonyságának a függvénye.

Összességében a fejezet mind architekturális, mind kísérleti bizonyítékot szolgáltat a tézis alátámasztására. A mobil eszközök beilleszthetők az elosztott gépi tanulási folyamatokba adat-párhuzamosság révén, amennyiben az adathalmazokat körültekintően particionálják, és megfelelő szinkronizációs mechanizmusokat alkalmaznak. Bár a teljesítményt továbbra is korlátozzák a jelenlegi szoftveres megvalósítások, a kísérletek egyértelműen igazolják, hogy kisebb adatrészletek megbízhatóan taníthatók mobil platformokon, így ezek a készülékek értékes és eddig alig kihasznált erőforrást jelentenek az elosztott tanulási rendszerekben.

Gépi tanulási könyvtárak összehasonlítása mobil eszközökön

2. Tézis: A mobil eszközökön végzett benchmark mérések megerősítik, hogy az eszközön történő tanítás megvalósíthatóságának értékelésében a CPU-használat és az energiahatékonyság a döntő tényezők; míg a Weka viszonylag alacsony erőforrás-felhasználást mutat, addig a SMILE CPU-intenzív működése korlátozza gyakorlati alkalmazhatóságát a gyengébb processzorokon.

A fejezet benchmarking vizsgálatai egyértelműen megmutatják, hogy a CPU-használat és az energiahatékonyság a legmeghatározóbb tényezők az eszközön történő gépi tanulási tanítás megvalósíthatóságának értékelésekor. A mobil eszközök erőforrásai jóval korlátozottabbak, mint az asztali vagy szerver környezetekben: a nagy számítási igényű folyamatok gyorsan kimeríthetik a rendelkezésre álló memóriát, túlterhelhetik a processzort, illetve jelentős akkumulátor-merülést okoznak. Emiatt a mobilon futtatható gépi tanulási könyvtárak értékelésekor nem elsősorban az algoritmikus teljesség, hanem a számítási és energiaigény kezelése válik döntő szemponttá.

A 2. és 3. táblázaton láthatók a könyvtárak megadott adathalmazra és algoritmusra vonatkozó eredményei. A prefix az adathalmazt jelöli (i az Iris adathalmazt, p a Patterns adathalmazt), a második rész pedig az algoritmus nevét (r = random forest, s = support-vector machine, k = K-means). A Patterns esetében a 6-os prefix 60 000 mintát jelent a teljes adathalmazból. A három vizsgált Java-alapú könyvtár közül – Weka, SMILE és Tribuo – a Weka mutatta a legalacsonyabb CPU-használatot és akkumulátor-fogyasztást. Az átlagos processzorterhelése jellemzően 20% körül maradt, így párhuzamosan más alkalmazások is működhettek a háttérben. Az energiaigénye a normalizált skálán alacsony tartományban mozgott (0,2–0,6 között). Fontos szempont, hogy a Weka nagy adathalmazok esetén is sikeresen lefutott, míg a SMILE és a Tribuo gyakran megszakadt a túlzott erőforrás-igény miatt. Ez a tulajdonság teszi a Wekát a legpraktikusabb választássá az eszközön történő tanításra, különösen közép- vagy gyengébb teljesítményű processzorok esetében.

2. táblázat - Algoritmusok CPU-használata a megadott adathalmazokon

	i-r	i-s	i-k	p-6-r	p-6-s	p-6-k	p-r	p-s	p-k
Tribuo	26	23	37	15	0	16	25	25	60
Weka	20	21	21	12	20	21	18	23	20
SMILE	97	71	16	90	30	21	75	30	61

3. táblázat - Algoritmusok energiafogyasztása a megadott adathalmazokon

	i-r	i-s	i-k	p-6-r	p-6-s	p-6-k	p-r	p-s	p-k
Tribuo	0.6	0.2	0.2	0.6	1	0.2	0.4	1	0.6
Weka	0.2	0.2	0.2	0.6	0.6	0.4	0.4	0.2	0.4
SMILE	0.6	0.2	0.6	1	0.6	0.4	1	0.6	0.2

Ezzel szemben a SMILE szinte minden esetben CPU-intenzívnek bizonyult, gyakran 90% feletti processzorhasználattal. Ez a mobil eszközökön a CPU teljes kihasználásához vezet, amely más alkalmazások akadozását, valamint az operációs rendszer általi leállítást is előidézheti. Energiafogyasztása szintén a legmagasabbak közé tartozott, több esetben elérte a maximális értéket. Bár bizonyos esetekben – például a K-means algoritmusnál – gyors futásidőt mutatott, a túlzott CPU- és akkumulátor-igény ellehetetleníti a gyakorlati alkalmazást. Ezek a jellemzők azt eredményezik, hogy a SMILE leginkább csak erős, csúcskategóriás készülékeken használható, szélesebb körű mobilalkalmazásokban viszont nem életképes.

A Tribuo a két szélsőség között helyezkedett el: bizonyos feladatokat megfelelő hatékonysággal tudott végrehajtani, de korlátozott kompatibilitása és valamivel nagyobb erőforrás-igénye a Wekához képest csökkenti a vonzerejét nagyobb vagy erőforrás-szűk környezetekben.

Összességében a benchmarking eredmények teljes mértékben alátámasztják a tézist: az eszközön történő tanítás gyakorlati megvalósíthatóságát elsődlegesen a CPU-használat és az energiafogyasztás határozza meg. A Weka alacsony erőforrás-igénye biztosítja a mobil alkalmazhatóságát, míg a SMILE processzorintenzív működése világosan megmutatja a könyvtár korlátait. Ez kiemeli annak fontosságát, hogy a gépi tanulási megoldásokat nemcsak algoritmikus erejük, hanem valós, erőforrás-korlátos környezetben mutatott hatékonyságuk alapján is értékelni kell.

Ensemble modellek integrálása mikroszolgáltatás-alapú architektúrákba

3. Tézis: Az ensemble gépi tanulási modellek mikroszolgáltatás-alapú architektúrákba való integrálása rámutat arra, hogy a különböző programozási nyelveken és keretrendszerekben fejlesztett heterogén

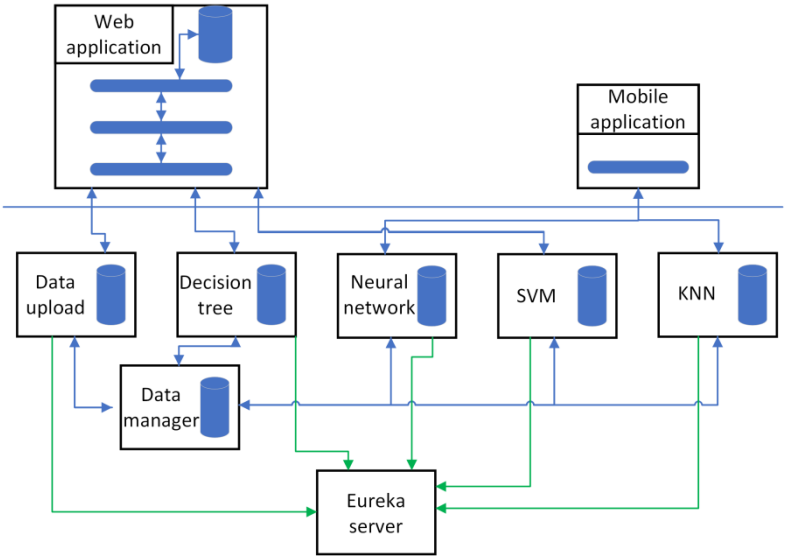
modellek webszolgáltatásokon keresztül egységesíthetők, ezáltal biztosítva a nyelvfüggetlenséget és a rendszerszintű interoperabilitást.

A tézis igazolását azok az architekturális és gyakorlati mechanizmusok adják, amelyek révén a gépi tanulási modellek becsomagolva és összehangolva működhetnek mikroszolgáltatásos környezetben. A gépi tanulási modellek jellemzően különböző programozási nyelveken és eltérő könyvtárak segítségével készülnek, ami közvetlen integrációjukat a nagyobb szoftverrendszerekbe megnehezíti. A modellek webszolgáltatásként való becsomagolása ezt a problémát úgy oldja meg, hogy egységes kommunikációs réteget biztosít. Az előrejelzések és a tanítási kérések REST-alapú interfészeken keresztül, JSON formátumban kerülnek továbbításra, így a kliensrendszer független marad a modellek belső reprezentációjától. Ez a megoldás ténylegesen megszünteti a nyelvi korlátokat: egy Java-ban implementált osztályozó és egy Pythonban megvalósított neurális háló egyaránt ugyanúgy érhető el, ami rendszerszinten biztosítja az egységesítést.

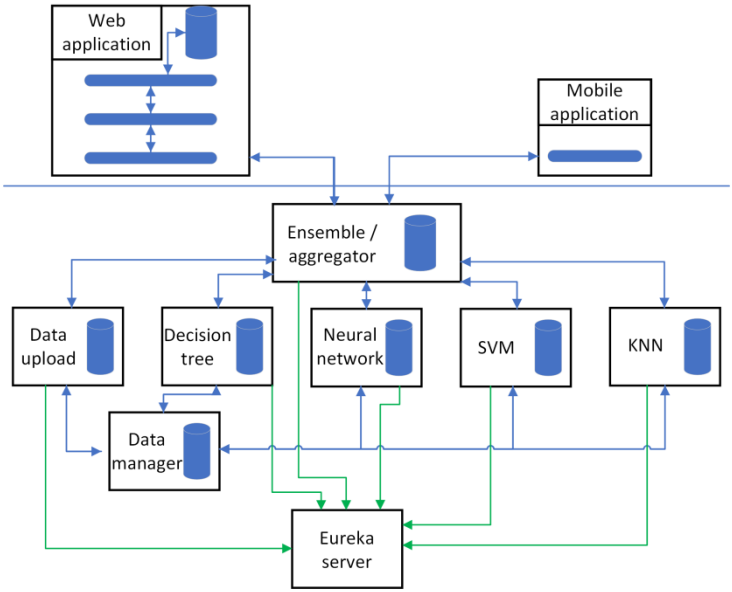
A mikroszolgáltatás-architektúra tovább erősíti ezt a függetlenséget. Minden modell önálló szolgáltatásként működik, saját adattárral és számítási logikával, a külvilág felé pedig csak egy publikus API-t biztosít. Ez a felépítés garantálja a laza csatolást, és lehetővé teszi a heterogén komponensek integrációját anélkül, hogy közös technológiai stack alkalmazására lenne szükség. Emellett skálázhatóságot és könnyű karbantarthatóságot biztosít, hiszen új modellek könnyen bevezethetők, a meglévők frissíthetők anélkül, hogy más részek működését zavarnák. Az interoperabilitás tehát nemcsak a technikai interfészek szintjén, hanem architekturális szinten is megvalósul: a szolgáltatások elosztva, egymástól függetlenül fejleszthetők és menedzselhetők, miközben egy közös rendszerben működnek együtt.

Az ensemble tanulás alkalmazása közvetlen bizonyítéka az egységesítés hatékonyságának. Az olyan technikák, mint a szavazás vagy a bagging, több modell előrejelzésének kombinációját igénylik, és az a tény, hogy ezek a módszerek különböző nyelvi környezetekben készült modellek között is sikeresen alkalmazhatók, azt mutatja, hogy a kimenetek egységesíthetők és kombinálhatók. Akár kliensoldalon, az egyes modellek eredményeinek közvetlen lekérdezésével, akár szerveroldalon, egy aggregátor mikroszolgáltatáson keresztül történik a kombináció, az ensemble konstrukció jól példázza, hogy a heterogén modellek együttesen is képesek egységes, közös előrejelzést nyújtani.

A gyakorlati megvalósítások megerősítik ennek kivitelezhetőségét. Mind a közvetlen modell-lekérdezésre épülő (3. ábra), mind a gateway-alapú architektúra (4. ábra) létrejött és tesztelésre került, és bebizonyosodott, hogy valós terhelés mellett is megbízhatóan kezelik a kéréseket és válaszokat (4. táblázat). Bár a gateway változat többlet késleltetést eredményez az extra kommunikációs lépések miatt, mindkét megoldás működőképes, és lehetőséget ad arra, hogy az alkalmazás igényei szerint mérlegeljük a teljesítménybeli kompromisszumokat. A választás a modellek számától, a klienseszközök számítási kapacitásától és a hálózati terhelés elosztásától függ, de mindkét esetben fennmarad a nyelvi függetlenség és az interoperabilitás alapelve.



3. ábra - Direkt architektúra-változat webes és mobil kliensekkel



4. ábra - Gateway architektúra-változat webes és mobil kliensekkel

4. táblázat – Közvetlen és Gateway variáns teljesítményei

Mutató	**Közvetlen változat**	**Gateway változat**
Virtuális felhasználók száma	20	20
Kiszolgált kérések száma (1 perc alatt)	1085	1004
Átlagos kérésszám (req/s)	13.71	12.96
Átlagos válaszidő (ms)	17	51
Minimum válaszidő (ms)	9	18
Maximum válaszidő (ms)	317	1221
90% válaszidő alatt (ms)	24	121
Hibaarány (%)	0	0

Összegzésként elmondható, hogy az ensemble gépi tanulási modellek mikroszolgáltatás-architektúrába való integrációja világosan megmutatja: a programozási nyelvek és keretrendszerek sokfélesége nem akadálya az egységes működésnek. A modellek webszolgáltatásokba történő absztrakciója és mikroszolgáltatási elvek mentén való összehangolása biztosítja a rugalmasságot és a megbízhatóságot. Az a képesség, hogy különböző forrásokból származó modellekből ensemble rendszerek építhetők, egyértelműen igazolja, hogy a nyelvi függetlenség és a rendszerinteroperabilitás nemcsak elérhető, hanem a gyakorlatban is megvalósítható, ezzel megerősítve a tézis állítását.

Számosság osztályozás az SMNIST kísérletekben

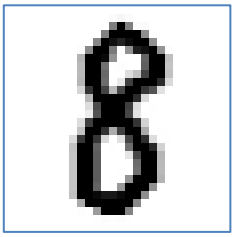
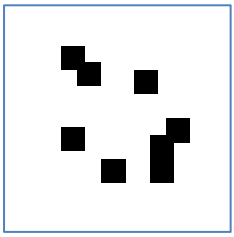
4. tézis: Az SMNIST kísérletekben a deeplearning4j LeNet MNIST modell teljesítménye – összhangban a többi vizsgált architektúrával – alátámasztja azt a megállapítást, hogy a mesterséges neurális hálók az emberekhez hasonlóan nagyobb pontossággal osztályozzák a számosságokat, ha azok a négyes OFS-kapacitás alatt maradnak.

Az 5.1-el jelzett fejezetben bemutatott SMNIST for Machines kísérletek alátámasztást adnak ahhoz a tézishez, hogy a számjegy- vagy képosztályozásra eredetileg kifejlesztett mélytanulási architektúrák az emberekéhez hasonló számosság-diszkriminációs mintázatot mutatnak. A legfontosabb megfigyelés, hogy a mesterséges neurális hálók (ANN-ek) nagy pontossággal ismerik fel a számosságokat akkor, amikor az elemek száma az emberi Object File System (OFS) határán belül marad, teljesítményük azonban romlik, ahogy a tárgyak száma növekszik.

Elsőként ismertetem a fogalmi hátteret. Az emberi kognícióban az OFS biztosítja, hogy gyorsan és pontosan tudjunk 3–4 tárgyig „szubitizálni”, vagyis azonnal felismerni a mennyiséget számlálás nélkül. Négy felett azonban a teljesítmény csökken, és az Approximate Number System

(ANS) lép működésbe, amely kevésbé pontos. A SMNIST for Humans kísérletek megerősítették ezt a jelenséget, kimutatva, hogy az emberek pontossága valóban visszaesik a négyes határ felett. A SMNIST for Machines vizsgálatok újdonsága abban rejlik, hogy megmutatták: az ANN-ek, explicit kognitív modell nélkül, hasonló teljesítménygörbét mutatnak.

A kísérletek módszertana szigorú volt. A SMNIST adatkészletek az MNIST formátumát követték, de a kézzel írt számjegyek helyett pontokból álló képeket tartalmaztak, legfeljebb kilenc darabig (5. ábra). Többféle változat (központosított vagy véletlen elhelyezés, különböző pixelek mérete, diszjunkt halmazok) gondoskodott arról, hogy a feladat ne legyen triviális, és ne lehessen egyszerű memorizációval megoldani. A kísérletekben standard architektúrákat (CNN, MLP, RNN) alkalmaztak anélkül, hogy azokat kifejezetten számosság-felismerésre alakították volna át. Így a teljesítmény valóban a modellek spontán generalizációs képességét tükrözte.


(a)

(b)

5. ábra - Tipikus bemeneti képek: (a) MNIST, (b) SMNIST

Az eredmények következetesek voltak: a kisebb számosságok esetén az architektúrák magas pontosságot értek el, de a pontosság fokozatosan és szisztematikusan csökkent a nagyobb számosságok (7–9 elem) esetében (5. táblázat). Míg például a CNN-ek és hierarchikus RNN-ek közel tökéletes teljesítményt mutattak négy elemig, addig nyolc vagy kilenc

elemnél az eredmények jelentősen visszaestek. Ez szorosan megfelelt az emberi OFS határának, és azt sugallta, hogy a hálók hasonló módon „reprezentálják” a kis számosságokat, mint ahogyan az emberi szubitizáció működik.

5. táblázat - „SMNIST for Machines” 2. sorozat mérései

Program	4 elem	5 elem	6 elem	7 elem	8 elem
Tensorflow 0.9.0, mnist_softmax.py	0.6317	0.3188	0.2334	0.1936	0.1658
Keras 2.2.4, mnist_cnn.py	0.9099	0.7055	0.7599	0.7285	0.6568
Keras/Hierarchical RNN	0.9993	0.9996	0.9442	0.9996	0.9993
PyTorch, cifar10_tutorial.py	0.8758	0.8589	0.723	0.556	0.6733
deeplearning4j LeNet MNIST	0.7743	0.5329	0.4770	0.3671	0.2977
Swift TF MNIST	0.6432	0.4906	0.3102	0.2896	0.1819
Swift TF CIFAR PyTorch	0.6729	0.6218	0.4796	0.4156	0.4802

A tapasztalt jelenség értelmezhető úgy, hogy a számosság-diszkrimináció az ANN-ekben nem előre tervezett, hanem a vizuális mintafelismerés általános mechanizmusaiból „emergens” módon jön létre. Ahogyan az emberek sem szimbolikus aritmetikára támaszkodnak 3–4 elemig, hanem egy alapvető perceptuális rendszerre, úgy a neurális hálók is természetes előnyt mutatnak a kis halmazok esetén, de

gyengébben teljesítenek a határ felett. Az emberi és mesterséges rendszerek teljesítménygörbéjének konvergenciája arra utal, hogy a mintázatfelismerés strukturális korlátai magukban hordozzák a szubitizáció-szerű működés lehetőségét.

A LeNet modell különösen jól szemlélteti ezt a tendenciát: a Series 2 adatkészleteiben 0,774-es pontosságot ért el a négy pontot tartalmazó képeken, azonban teljesítménye öt pontnál már 0,532-re, hat pontnál 0,477-re, hét pontnál 0,367-re, nyolc pontnál pedig mindössze 0,297-re esett vissza. Az „SMNIST for Anyone” sorozatban, ahol a pontokat egyszerű szimbólumokkal helyettesítették, ugyanez a mintázat volt megfigyelhető: viszonylag magas pontosság négy objektumnál, majd meredek teljesítménycsökkenés a nagyobb számosságok esetén.

Irodalomjegyzék

[1] K. Kourou, T. P. Exarchos, K. P. Exarchos, M. V. Karamouzis, and D. I. Fotiadis, “Machine learning applications in cancer prognosis and prediction,” Computational and Structural Biotechnology Journal, vol. 13, pp. 8–17, Nov. 2014, doi: 10.1016/j.csbj.2014.11.005.

[2] M. Kukar, I. Kononenko, C. Grošelj, K. Kralj, and J. Fettich, “Analysing and improving the diagnosis of ischaemic heart disease with machine learning,” Artificial Intelligence in Medicine, vol. 16, no. 1, pp. 25–50, May 1999, doi: 10.1016/s0933-3657(98)00063-3.

[3] S. Li et al., “Hippocampal shape analysis of Alzheimer disease based on machine learning methods,” American Journal of Neuroradiology, vol. 28, no. 7, pp. 1339–1345, Aug. 2007, doi: 10.3174/ajnr.a0620.

[4] M. Chen, Y. Hao, K. Hwang, L. Wang, and L. Wang, “Disease prediction by machine learning over big data from healthcare communities,” IEEE Access, vol. 5, pp. 8869–8879, Jan. 2017, doi: 10.1109/access.2017.2694446.

[5] M. S. Bartlett, G. Littlewort, M. Frank, C. Lainscsek, I. Fasel, and J. Movellan, “Recognizing Facial Expression: Machine Learning and Application to Spontaneous Behavior,” 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05), vol. 2. IEEE, pp. 568–573. doi: 10.1109/cvpr.2005.297.

[6] S. Zander, T. Nguyen, and G. Armitage, “Automated traffic classification and application identification using machine learning,” The IEEE Conference on Local Computer Networks 30th Anniversary (LCN’05)l. IEEE, pp. 250–257, 2005. doi: 10.1109/lcn.2005.35.

[7] K. Alsabti, S. Ranka, and V. Singh, “CLOUDS: a decision tree classifier for large datasets,” Knowledge Discovery and Data Mining, pp. 2–8, Aug. 1998, [Online]. Available: https://aaaipress.org/Papers/KDD/1998/KDD98-001.pdf

[8] S.-M. Lee and P. A. Abbott, “Bayesian networks for knowledge discovery in large datasets: basics for nurse researchers,” Journal of Biomedical Informatics, vol. 36, no. 4–5, pp. 389–399, Aug. 2003, doi: 10.1016/j.jbi.2003.09.022.

[9] H. Yu, J. Yang, and J. Han, “Classifying large data sets using SVMs with hierarchical clusters,” Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp. 306–315, Aug. 24, 2003. doi: 10.1145/956750.956786.

[10] “PassMark Android Benchmark Charts - CPU Rating.” https://www.androidbenchmark.net/cpumark_chart.html, (last visited 24 August 2025).

[11] “Smartphone Processors - Benchmark List,” Notebookcheck, May 20, 2019. https://www.notebookcheck.net/Smartphone-Processors-Benchmark-List.149513.0.html, (last visited 24 August 2025).

[12] Statista, “Share of global mobile website traffic 2015-2024,” Statista, Jan. 28, 2025. https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobiledevices/, (last visited 24 August 2025).

[13] Statista, “U.S. daily media usage time via mobile 2019-2023,” Statista, Jul. 08, 2025. https://www.statista.com/statistics/469983/time-spent-mobile-media-type-usa, (last visited 24 August 2025).

[14] “Mobile vs. Desktop Usage in 2020,” Perficient. https://www.perficient.com/insights/research-hub/mobile-vs-desktop-usage, (last visited 24 August 2025).

[15] G. Seni and J. F. Elder, “Ensemble methods in data mining: Improving accuracy through combining predictions,” Synthesis Lectures on Data Mining and Knowledge Discovery, vol. 2, no. 1, pp. 1–126, Jan. 2010, doi: 10.2200/s00240ed1v01y200912dmk002.

[16] A. Guazzelli, W.-C. Lin, and T. Jena, PMML in action: unleashing the power of open standards for data mining and predictive analytics. Seattle: CreateSpace, 2010.

[17] W.-F. Lin et al., “ONNC: A Compilation Framework Connecting ONNX to Proprietary Deep Learning Accelerators,” 2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS). IEEE, pp. 214–218, Mar. 2019. doi: 10.1109/aicas.2019.8771510.

[18] Onnx, “GitHub - onnx/onnx: Open standard for machine learning interoperability,” GitHub. https://github.com/onnx/onnx, (last visited 24 August 2025).

[19] B. Amos, H. Turner, and J. White, “Applying machine learning classifiers to dynamic Android malware detection at scale,” 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC). IEEE, pp. 1666–1671, July 2013. doi: 10.1109/iwcmc.2013.6583806.

[20] N. Peiravian and X. Zhu, “Machine Learning for Android Malware Detection Using Permission and API Calls,” 2013 IEEE 25th International Conference on Tools with Artificial Intelligence. IEEE, pp. 300–305, Nov. 2013. doi: 10.1109/ictai.2013.53.

[21] J. Sahs and L. Khan, “A Machine Learning Approach to Android Malware Detection,” 2012 European Intelligence and Security Informatics Conference. IEEE, pp. 141–147, Aug. 2012. doi: 10.1109/eisic.2012.34.

[22] Ž. Jovanović, D. Jagodić, D. Vujičić, and S. Ranđić, “Java Spring Boot REST Web Service Integration with Java Artificial Intelligence Weka Framework,” in Proc. International Scientific Conference UNITEH 2017, Gabrovo, Bulgaria, Nov. 17–18, 2017, pp. 270–274. [Online]. Available: https://www.unitechsp.tugab.bg/images/papers/2017/s5/s5_p238.pdf

[23] M. Li et al., “Scaling Distributed Machine Learning with the Parameter Server,” in Proc. 11th USENIX Symposium on Operating Systems Design and Implementation. (OSDI ’14), Broomfield, CO, USA, Oct. 6–8, 2014, pp. 583–598.

[24] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan, “MLbase: A Distributed Machine-learning System,” in Proc. 6th

Biennial Conference on Innovative Data Systems Research (CIDR ’13), Asilomar, CA, USA, Jan. 2013.

[25] E. R. Sparks et al., “MLI: An API for Distributed Machine Learning,” 2013 IEEE 13th International Conference on Data Mining. IEEE, pp. 1187–1192, Dec. 2013. doi: 10.1109/icdm.2013.158.

[26] Meng, X., et al. "MLlib: machine learning in apache spark," in The Journal of Machine Learning Research, vol. 17, no. 1, pp. 1235–1241, 2016.

[27] A. McIntosh, S. Hassan, and A. Hindle, “What can Android mobile app developers do about the energy consumption of machine learning?,” Empirical Software Engineering, vol. 24, no. 2, pp. 562–601, June 2018, doi: 10.1007/s10664-018-9629-2.

[28] A. Ignatov et al., “AI Benchmark: Running Deep Neural Networks on Android Smartphones,” Lecture Notes in Computer Science. Springer International Publishing, pp. 288–314, 2019. doi: 10.1007/978-3-030-11021-5_19.

[29] A. Kulkarni, N. Mhalgi, S. Gurnani, and N. Giri, “Pothole Detection System using Machine Learning on Android,” International Journal of Emerging Technology and Advanced Engineering, vol. 4, no. 7, pp. 360–364, Jul. 2014. [Online]. Available: http://www.ijetae.com/files/Volume4Issue7/IJETAE_0714_55.pdf

[30] R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore, “PMLB: a large benchmark suite for machine learning evaluation and comparison,” BioData Mining, vol. 10, no. 1, Dec. 2017, doi: 10.1186/s13040-017-0154-4.

[31] C. Coleman et al., “Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark,” ACM SIGOPS Operating Systems Review, vol. 53, no. 1, pp. 14–25, July 2019, doi: 10.1145/3352020.3352024.

[32] P. Mattson, C. Cheng, C. Coleman, G. Diamos, P. Micikevicius, D. Patterson, et al., “MLPerf Training Benchmark,” in Proceedings of Machine Learning and Systems, vol. 2, I. Dhillon, D. Papailiopoulos, and V. Sze, Eds.,

2020, pp. 336–349. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2020/file/411e39b117e885341f25efb8912945f7-Paper.pdf

[33] F. Pirotti, F. Sunar, and M. Piragnolo, “BENCHMARK OF MACHINE LEARNING METHODS FOR CLASSIFICATION OF A SENTINEL-2 IMAGE,” International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences, vol. XLI-B7, pp. 335–340, June 2016, doi: 10.5194/isprsarchives-xli-b7-335-2016.

[34] E. Frank, M. A. Hall, and I. H. Witten, “The WEKA Workbench: Online Appendix for Data Mining: Practical Machine Learning Tools and Techniques,” Morgan Kaufmann, 4th ed., Online appendix, 2016. [Online]. Available: https://ml.cms.waikato.ac.nz/weka/Witten_et_al_2016_appendix.pdf.

[35] H. Li, “Smile - home.” https://haifengl.github.io/. (Last visited 24 August 2025)

[36] Oracle, Tribuo: Machine Learning in Java. https://tribuo.org/ (Last visited 24 August 2025)

[37] T. Chen and C. Guestrin, “XGBoost,” Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, pp. 785–794, Aug. 13, 2016. doi: 10.1145/2939672.2939785.

[38] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, et al., “TensorFlow: A System for Large-Scale Machine Learning,” in Proceedings of USENIX Symposium on Operating Systems Design and Implementation. (OSDI ’16), Savannah, GA, USA, Nov. 2–4, 2016, pp. 265–283.

[39] Deeplearning4j Development Team, “Deeplearning4j: Open-source Distributed Deep Learning for the JVM,” 2016. GitHub. http://deeplearning4j.org/. (Last visited 24 August 2025)

[40] H2oai, GitHub - h2oai/h2o-3. https://github.com/h2oai/h2o-3

[41] P. K. Falk, “Tech Services on the Web: MALLET-MAchine Learning for LanguagE Toolkit; http://mallet.cs.umass.edu/,” Technical Services

Quarterly, vol. 31, no. 4, pp. 410–411, Sept. 2014, doi: 10.1080/07317131.2014.943038.

[42] M.-O. Pahl and M. Loipfinger, “Machine learning as a reusable microservice,” NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium. IEEE, pp. 1–7, Apr. 2018. doi: 10.1109/noms.2018.8406165.

[43] Y.-D. Bromberg and L. Gitzinger, “DroidAutoML: A Microservice Architecture to Automate the Evaluation of Android Machine Learning Detection Systems,” Lecture Notes in Computer Science. Springer International Publishing, pp. 148–165, 2020. doi: 10.1007/978-3-030-50323-9_10.

[44] J. L. Ribeiro, M. Figueredo, A. Araujo, N. Cacho, and F. Lopes, “A Microservice Based Architecture Topology for Machine Learning Deployment,” 2019 IEEE International Smart Cities Conference (ISC2). IEEE, pp. 426–431, Oct. 2019. doi: 10.1109/isc246665.2019.9071708.

[45] D. C. Attota, V. Mothukuri, R. M. Parizi, and S. Pouriyeh, “An Ensemble Multi-View Federated Learning Intrusion Detection for IoT,” IEEE Access, vol. 9, pp. 117734–117745, 2021, doi: 10.1109/access.2021.3107337.

[46] P. Salza, E. Hemberg, F. Ferrucci, and U.-M. O’Reilly, “cCube,” Proceedings of the Genetic and Evolutionary Computation Conference Companion. ACM, pp. 137–138, July 15, 2017. doi: 10.1145/3067695.3076089.

[47] J. Gruendner et al., “KETOS: Clinical decision support and machine learning as a service – A training and deployment platform based on Docker, OMOP-CDM, and FHIR Web Services,” PLoS ONE, vol. 14, no. 10, p. e0223010, Oct. 2019, doi: 10.1371/journal.pone.0223010.

[48] M. Chippa, A. Priyadarshini, and R. Mohanty, “Application of Machine Learning Techniques to Classify Web Services,” 2019 IEEE International Conference on Intelligent Techniques in Control, Optimization and Signal Processing (INCOS). IEEE, pp. 1–7, Apr. 2019. doi: 10.1109/incos45849.2019.8951339.

[49] H. Alipour and Y. Liu, “Online machine learning for cloud resource provisioning of microservice backend systems,” 2017 IEEE International Conference on Big Data (Big Data). IEEE, pp. 2433–2441, Dec. 2017. doi: 10.1109/bigdata.2017.8258201.

[50] H. Chang, M. Kodialam, T. V. Lakshman, and S. Mukherjee, “Microservice Fingerprinting and Classification using Machine Learning,” 2019 IEEE 27th International Conference on Network Protocols (ICNP). IEEE, pp. 1–11, Oct. 2019. doi: 10.1109/icnp.2019.8888077.

[51] H. Harms, C. Rogowski, and L. Lo Iacono, “Guidelines for adopting frontend architectures and patterns in microservices-based systems,” Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. ACM, pp. 902–907, Aug. 21, 2017. doi: 10.1145/3106237.3117775.

[52] D. Taibi, V. Lenarduzzi, C. Pahl, and A. Janes, “Microservices in agile software development,” Proceedings of the XP2017 Scientific Workshops. ACM, pp. 1–5, May 22, 2017. doi: 10.1145/3120459.3120483.

[53] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, “A Survey on Distributed Machine Learning,” ACM Computing Surveys, vol. 53, no. 2, pp. 1–33, Mar. 2020, doi: 10.1145/3377454.

[54] J. A. Valdivia, A. Lora-González, X. Limón, K. Cortes-Verdin, and J. O. Ocharán-Hernández, “Patterns Related to Microservice Architecture: a Multivocal Literature Review,” Programming and Computer Software, vol. 46, no. 8, pp. 594–608, Dec. 2020, doi: 10.1134/s0361768820080253.

[55] H. Chawla and H. Kathuria, “Implementing Microservices,” Building Microservices Applications on Microsoft Azure. Apress, pp. 21–41, 2019. doi: 10.1007/978-1-4842-4828-7_2.

[56] K. S. Prasad Reddy, Beginning Spring Boot 2. Apress, 2017. doi: 10.1007/978-1-4842-2931-6.

[57] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, et al., “Scikit-learn: Machine Learning in Python,” Journal of Machine

Learning Research, vol. 12, pp. 2825–2830, Oct. 2011. doi: 10.48550/arXiv.1201.0490.

[58] M. Grinberg, Flask web development: developing web applications with Python, 1. ed. Beijing Köln: O'Reilly, 2014.

Releváns publikációk listája

Referált folyóiratcikkek

[J1] N. Bátfai et al., “Object file system software experiments about the notion of number in humans and machines,” Cognition Brain Behavior an Interdisciplinary Journal, vol. 23, no. 4, pp. 257–280, Dec. 2019, doi: 10.24193/cbb.2019.23.15.

[J2] M. Szabó, “Distributed Machine Learning Using Data Parallelism on Mobile Platform,” Journal of Mobile Multimedia, vol. 16, no. 3, pp. 317-334, Sept. 2020, doi: 10.13052/jmm1550-4646.1633.

[J3] M. Szabó, “Building Ensemble Models with Web Services on Microservice Architecture,” Informatica, vol. 48, no. 7, pp. 1-10, Mar. 2024, doi: 10.31449/inf.v48i7.4918.

Konferenciakötetben megjelent cikkek

[I1] M. Szabó, ”Machine Learning on Android with Oracle Tribuo, SMILE and Weka,” CEUR Workshop Proceedings, vol. 2874, pp. 176-186, 2021.

[I2] N. Bátfai, T. Tutor, Z. Bartha, A. Czanik, M. Szabó, ”Red Flower Hell: a Minecraft MALMÖ Challenge to Support Introductory Programming Courses,” CEUR Workshop Proceedings, vol. 2874, pp. 56-66, 2021.

DEBRECENI EGYETEM
EGYETEMI ÉS NEMZETI KÖNYVTÁR
H-4002 Debrecen, Egyetem tér 1, Pf.: 400
Tel.: 52/410-443, e-mail: publikaciok@lib.unideb.hu

Nyilvántartási szám: DEENK/494/2025.PL
Tárgy: PhD Publikációs Lista

Jelölt: Szabó Máté
Doktori Iskola: Informatikai Tudományok Doktori Iskola
MTMT azonosító: 10071945

A PhD értekezés alapjául szolgáló közlemények

Idegen nyelvű tudományos közlemények külföldi folyóiratban (3)

1. **Szabó, M.**: Building Ensemble Models with Web Services on Microservice Architecture.
   Informatica (Slovenia). 48 (7), 1-10, 2024. ISSN: 0350-5596.
   DOI: http://dx.doi.org/10.31449/inf.v48i7.4918

2. **Szabó, M.**: Distributed Machine Learning Using Data Parallelism on Mobile Platform.
   JMM. 16 (3), 317-334, 2020. ISSN: 1550-4646.
   DOI: http://dx.doi.org/10.13052/jmm1550-4646.1633

3. Bátfai, N., Papp, D., Bogacsovics, G., **Szabó, M.**, Simkó, V. S., Bersenszki, M., Szabó, G., Kovács, L., Kovács, F., Varga, E. S.: Object file system software experiments about the notion of number in humans and machines.
   Cognition, Brain, Behavior. 23 (4), 257-280, 2019. ISSN: 2247-9228.
   DOI: http://dx.doi.org/10.24193/cbb.2019.23.15

Idegen nyelvű konferencia közlemények (2)

4. **Szabó, M.**: Machine Learning on Android with Oracle Tribuo, SMILE and Weka.
   In: Proceedings of the 1st Conference on Information Technology and Data Science. Ed.: István Fazekas, András Hajdu, Tibor Tómács, CEUR Workshop Proceedings, Debrecen, 176-186, 2021, (CEUR Workshop Proceedings, ISSN 1613-0073 ; 2874.)

5. Bátfai, N., Tutor, T., Bartha, Z., Czanik, A., **Szabó, M.**: Red Flower Hell: a Minecraft MALMÖ Challenge to Support Introductory Programming Courses.
   In: Proceedings of the 1st Conference on Information Technology and Data Science. Ed.: István Fazekas, András Hajdu, Tibor Tómács, CEUR Workshop Proceedings, Debrecen, 56-66, 2021, (CEUR Workshop Proceedings, ISSN 1613-0073 ; 2874.)

DEBRECENI EGYETEM
EGYETEMI ÉS NEMZETI KÖNYVTÁR
H-4002 Debrecen, Egyetem tér 1, Pf.: 400
Tel.: 52/410-443, e-mail: publikaciok@lib.unideb.hu

További közlemények

Idegen nyelvű tudományos közlemények hazai folyóiratban (1)

6. Bátfai, N., Besenczi, R., Jeszenszky, P., **Szabó, M.**, Ispány, M.: Markov modeling of traffic flow in Smart Cities.
   Ann. Math. Inform. 53, 21-44, 2021. ISSN: 1787-5021.
   DOI: http://dx.doi.org/10.33039/ami.2021.04.008

Idegen nyelvű konferencia közlemények (2)

7. Jeszenszky, P., Besenczi, R., **Szabó, M.**, Ispány, M.: Estimating road traffic flows in macroscopic Markov model.
   In: 2022 IEEE 2nd Conference on Information Technology and Data Science (CITDS), IEEE, Piscataway, 136-141, 2022. ISBN: 9781665496537
8. Bátfai, N., **Szabó, M.**: Possible Neural Models to Support the Design of Prime Convo Assistant.
   In: Proceedings of the 1st Conference on Information Technology and Data Science. Ed.: István Fazekas, András Hajdu, Tibor Tómács, CEUR Workshop Proceedings, Debrecen, 46-55, 2021, (CEUR Workshop Proceedings, ISSN 1613-0073 ; 2874.)

A DEENK a Jelölt által a Tudóstérbe feltöltött adatok bibliográfiai és tudománymetriai ellenőrzését a tudományos adatbázisok és a Journal Citation Reports Impact Factor lista alapján elvégezte.

Debrecen, 2025.08.27.

DEBRECENI EGYETEM * Egyetemi és Nemzeti Könyvtár *

Short thesis for the degree of doctor of philosophy (PhD)

Design and performance analysis of applications using machine learning

by Máté Szabó

Supervisor: Dr. Márton Ispány

UNIVERSITY OF DEBRECEN

Doctoral School of Informatics

Debrecen, 2025

Contents

Introduction and motivation

Background

Data is the foundation of both science and everyday life, and with technological progress it has become a key resource of the 21st century. Machine learning enables the interpretation of such data in three main forms: supervised, unsupervised, and semi-supervised learning.In software development, the rise of web and mobile applications provides platform-independent, easily updatable, and personalized solutions.The dissertation focuses on the integration of machine learning with web and mobile applications. Through case studies, it demonstrates how intelligent, scalable, and user-friendly systems can be built, showing that machine learning can be embedded into virtually any software connected to the Internet.

Machine Learning on Smartphones: Processing Large Datasets in a Distributed Architecture

Machine learning today appears in nearly all digital services, with applications ranging from healthcare and autonomous driving to commerce and political analysis. In medicine, it is used for cancer prediction [1], improving cardiac diagnostics [2], analyzing Alzheimer-related changes [3], and early disease detection [4]. Other examples include facial expression recognition [5] and Internet traffic classification [6].
The ever-growing data volume demands more efficient approaches. Traditional methods – such as decision trees [7], Bayesian networks [8], or support vector machines [9] – remain useful but face limitations with large-scale datasets.

Mobile phones have become the most widespread computing devices, with rapidly improving hardware [10, 11]. Many high-end models can

already run ML models locally, yet their resources often remain underutilized [12–14].

The prototype I developed consists of two main parts:

- **Web service**: receives and stores models from mobile devices, builds ensemble models (improving prediction accuracy [15]), and manages large datasets.
- **Mobile application**: trains simple neural networks, uses pre-trained models, and communicates with the central server.

This work connects to mobile and distributed ML, as well as model exchange formats:

- **Model exchange**: e.g., PMML [16] and ONNX [17, 18].
- **Mobile ML**: image recognition, malware detection on Android [19–21].
- **ML via web services**: access to pre-trained models for limited devices [22].
- **Distributed ML**: parameter-server approaches [23], and frameworks like MLBase [24], MLI API [25], and Apache Spark MLlib [26].

Machine Learning on Mobile Devices and Java-Based Solutions

With the advancement of mobile devices, machine learning (ML) has become essential for personalized user experiences. Most applications rely on pre-trained models (e.g., speech recognition, image enhancement), while on-device training faces challenges such as high energy consumption [27]. Still, useful applications exist, such as benchmarking mobile processors [28], pothole detection [29], or Android malware detection [21].

Chapter 3 of the dissertation focuses on the intersection of ML and Android benchmarking. Several benchmark systems are discussed, including PMLB [30], DAWNBench [31], MLPerf [32], and Sentinel-2 image classification [33].

Although Java is not a dominant ML language, it remains central in application development, with several available libraries:

- **Weka** [34]: full-featured ML software with GUI and library support.
- **SMILE** [35]: high-performance engine covering a broad algorithmic spectrum.
- **Oracle Tribuo** [36]: offers provenance, type safety, interoperability; integrates with XGBoost [37], TensorFlow [38], and ONNX [18].
- **Deeplearning4J** [39]: Java-based library focused on deep learning.
- **H2O** [40]: in-memory platform with REST API integration.
- **Mallet** [41]: specialized in NLP, document classification, and clustering.

Integration of Microservices and Machine Learning

Numerous studies address the intersection of machine learning (ML) and microservices. Some define ML as a service exposed via REST interfaces [42], while others adapt this approach to mobile platforms, such as DroidAutoML for Android malware detection [43]. More general ML-supporting architectures have also been proposed [44], as well as ensemble-based systems combining multiple models and API gateways [45]. The cCube architecture [46] employs a central orchestrator, similar to the approach followed in this research.

Applications include healthcare (e.g., the KETOS decision-support system [47]) and software engineering, where ML can classify microservices or allocate resources [48–50]. Architectural design is supported by frontend guidelines [51] and common patterns such as API gateway and aggregator [54, 55].

Despite their advantages, microservices increase system complexity, introducing challenges in versioning, monitoring, and state management [52]. Distributed ML further raises computation time, energy demand, and the difficulty of managing parallel resources [53].

We identified two models for the integration of machine learning software components:

- **Gateway model**: results are aggregated by a central service.
- **Direct model**: the client receives partial results and performs the aggregation itself.

Since a shared database for such software components would violate the independence defined as one of the advantages of microservice architectures, each service uses its own database.

Our reference implementation was built with Spring Boot and Spring Cloud, using Eureka Server for service discovery [56]. Java-based services employ Weka [34] and Deeplearning4j [38], while Python-based ones (Scikit-learn [57], TensorFlow [39], Azure ML [58]) use the Flask framework.

Our Research

I began my research work in 2018, pursuing two threads simultaneously, as I felt they would eventually converge towards the end of my studies. One aim was to experiment with machine learning algorithms on various applications and data. The other was to develop a more general framework or reference implementation that would allow

the computational capacities of simpler devices to be coordinated for model training, preferably in a platform-independent manner.

The first experimental project was SMNIST, where we compared numerosity recognition in human and machine environments. As several of us were examining the same datasets of varying difficulty, I had the opportunity to become acquainted with multiple platforms and tools, which were predominantly used for training convolutional neural networks. A problem already emerged here: since everyone trained their models in their own environments using different languages, these could not be easily implemented for another task, meaning there was no unified accessibility.

As a first step toward unified accessibility, I studied the concepts of model and data parallelism. Initially, I implemented a data-parallel solution, using a semi-independent approach with the Java language and the Android platform. The goal here was to slice a larger dataset into smaller models in both desktop and mobile environments, and then manage them in a shared location. The solution to this was bytecode serialization, which wrote the trained models into a byte array and then reassembled them in a Java environment, enabling parameter management in one place or encapsulation into an ensemble model.

As machine learning platforms became more extensive, allowing existing algorithms to be tested in new, more modern environments. This led to the "Red Flower Hell" project, one of its aims being to foster a demand for machine learning tools during education. The environment here was Microsoft's Minecraft Malmö, the essence of which is that an agent has to be programmed to solve a specific task within the Minecraft game. This was an excellent training ground for students learning high-level programming, who, using this tool, attempted to compete with each other using simple algorithms in

Python. Since this task cannot be solved effectively with traditional algorithms, becoming familiar with AI tools became essential.

After the mobile platform–based data-parallel solution, I wanted my framework to natively support the most efficient machine learning library for mobile devices. Therefore, I evaluated the three most popular ones—Oracle Tribuo, Weka, and SMILE—based on four criteria. For the experiments, I chose datasets of different sizes and complexities and tested the same algorithms across the three libraries. The results were mixed in terms of runtime, memory requirements, battery usage, and average CPU load, but overall, Weka—which I had also used in earlier work—performed the best.

Since the previous work was heavily Java-based, I needed to move further toward platform and language independence. This required abandoning the concept of managing all models in a single shared system. The new direction was a more loosely coupled, easily extensible solution, inspired by microservices. By nature, microservices are written in different languages, run on different platforms, and rely on different data sources, so it was no longer necessary to convert models into a common format. The concept here is that trained models are developed in various environments and then published as microservices, so any other software within the system can use them. Instead of merging multiple models, ensemble models can be employed to combine the existing ones. The main challenges in this approach concern the accessibility of data and models, as well as the distribution of computational load. By fine-tuning these aspects, the system can be tailored to specific needs, whether the load lies on the server side or the client side.

New scientific results and theses of the dissertation

Data-Parallel Machine Learning on Mobile Devices

Thesis 1: The integration of mobile devices into distributed machine learning through data parallelism is technically feasible. Although the overall performance is largely dependent on the implementation of the utilized libraries, measurements demonstrate that the processing of smaller dataset partitions can be executed reliably and effectively on mobile platforms.

The feasibility of integrating mobile devices into distributed machine learning via data parallelism is demonstrated in Chapter 2. This presents a working architecture that combines a server-side web service, a relational database, and Android-based clients (Figure 1). Within this framework, mobile devices can request dataset slices, train models on these smaller partitions, and then return the trained models to the server for aggregation. This architecture ensures that mobile devices, despite their limited resources, are capable of contributing to distributed training processes in a technically reliable way.

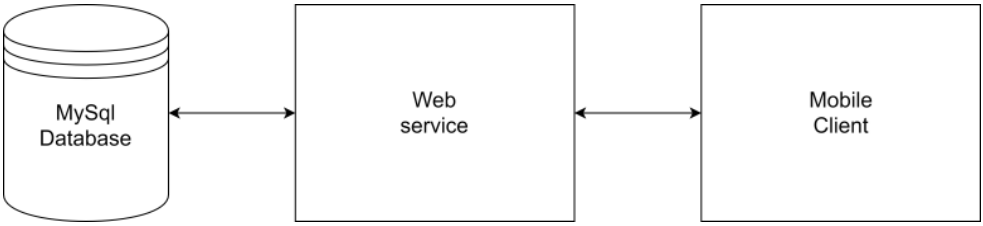
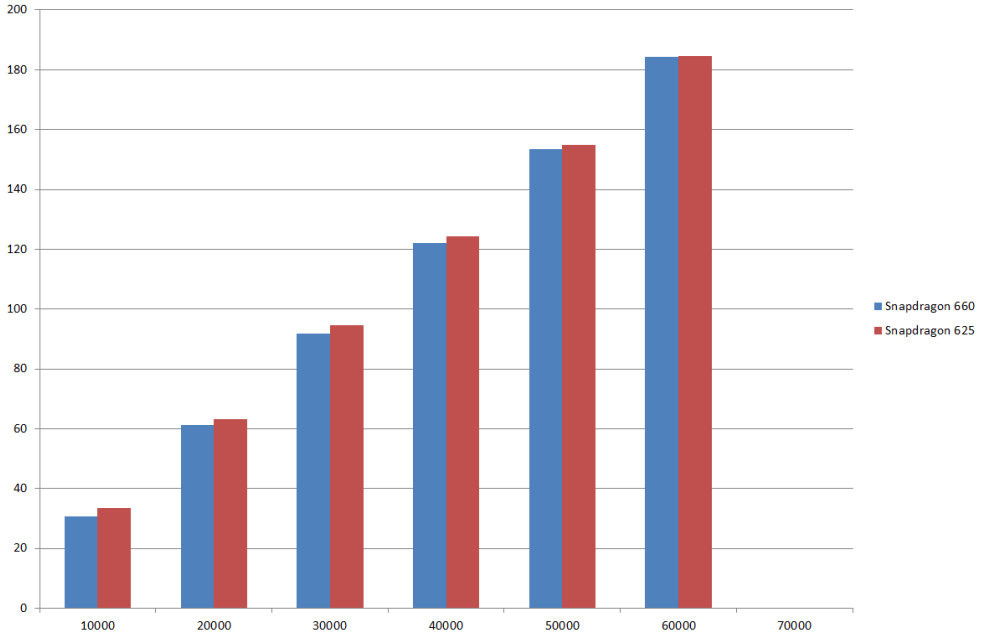


Figure 1. - The relationship of the system's components

The results show that the system operates correctly and produces meaningful models. By applying data slicing, even large datasets, such as the Record Linkage Comparison Patterns dataset with millions of

records, could be partially processed on mobile devices. While desktop systems are naturally more powerful, the measurements confirmed that smaller dataset partitions—up to roughly 70,000 records—can be trained successfully on smartphones (Figure 2). Moreover, the evaluation of models showed that the accuracy, precision and recall values of mobile-trained submodels were comparable to those produced on desktops (Table 1). This points out that the approach is not only feasible but can also be effective in certain cases, since mobile sub-models can be aggregated into larger ensemble models, while the predictive performance may remain the same.



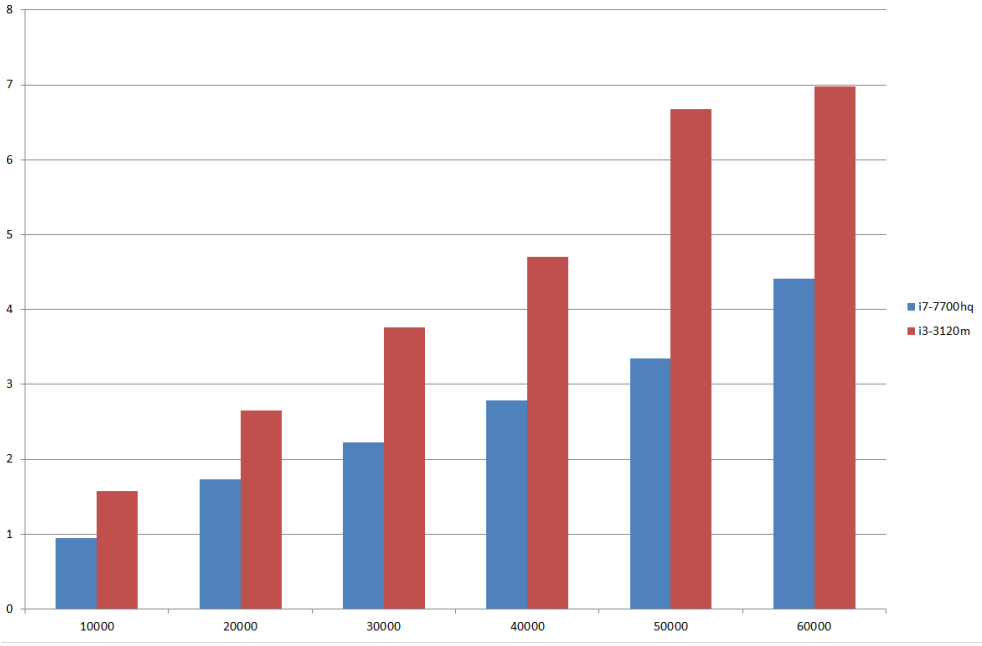


Figure 2. - Runtime of 1 epoch based on the size of the dataset on a) mobile and b) desktop

Table 1. - Accuracy and best result in mobile and desktop environments

environment	accuracy	score at the best epoch
mobile	0.7907	4.81924921875
computer	0.7907	4.81931338500

At the same time, the chapter highlights that performance is highly dependent on the chosen machine learning libraries. Deeplearning4J was used because of its compatibility across server and Android, but its limitations on mobile devices became evident. The lack of thread-safety and the inability to fully exploit multicore architectures resulted in relatively slow training times. Thus, the overall performance of the

system is less a question of the raw hardware potential of smartphones, and more of the efficiency of the libraries that enable learning on them.

Altogether, the chapter provides both architectural and experimental evidence that supports the thesis. Mobile devices can be integrated into distributed machine learning workflows through data parallelism, provided that datasets are carefully partitioned and suitable synchronization mechanisms are applied. While performance remains constrained by current software implementations, the experiments confirm that smaller dataset partitions can be trained reliably on mobile platforms, making them a valuable and underutilized resource in distributed learning systems.

Comparison of Machine Learning Libraries on Mobile Devices

Thesis 2: Benchmarking on mobile devices confirms that CPU usage and energy efficiency are decisive factors in assessing the feasibility of on-device training; while Weka maintains relatively low consumption, SMILE proves to be CPU-intensive, limiting its practicality for weaker processors

The third chapter clearly demonstrates that CPU usage and energy efficiency are the decisive factors when evaluating the feasibility of on-device machine learning training. Mobile devices have limited resources compared to desktop or server environments, and intensive training processes not only risk exhausting available memory but also generate high processor loads and rapid battery drain. These constraints mean that the viability of any machine learning library on Android depends less on its algorithmic completeness and more on its efficiency in handling computational and energy demands.

Tables 2 and 3 present the results of the libraries on the specified datasets and algorithms. The prefix indicates the dataset (i for the Iris

dataset, p for the Patterns dataset), while the second part denotes the name of the algorithm (r = random forest, s = support-vector machine, k = K-means). In the case of the Patterns dataset, the prefix 6 represents 60,000 samples from the full dataset. Among the three tested Java-based libraries—Weka, SMILE, and Tribuo—Weka consistently showed the lowest CPU usage and the lowest battery consumption. Its average CPU load typically stayed around 20%, leaving enough headroom for parallel applications to run, while its normalized battery consumption remained low (mostly between 0.2 and 0.6). Importantly, Weka was also able to complete training on very large datasets, where SMILE and Tribuo often failed due to excessive resource needs. These properties make Weka the most practical choice for on-device training, especially on mid-range or weaker processors where efficiency is critical.

Table 2. - CPU usage of algorithms on the specified datasets

	i-r	i-s	i-k	p-6-r	p-6-s	p-6-k	p-r	p-s	p-k
Tribuo	26	23	37	15	0	16	25	25	60
Weka	20	21	21	12	20	21	18	23	20
SMILE	97	71	16	90	30	21	75	30	61

Table 3. - Energy consumption of algorithms on the specified datasets

	i-r	i-s	i-k	p-6-r	p-6-s	p-6-k	p-r	p-s	p-k
Tribuo	0.6	0.2	0.2	0.6	1	0.2	0.4	1	0.6
Weka	0.2	0.2	0.2	0.6	0.6	0.4	0.4	0.2	0.4
SMILE	0.6	0.2	0.6	1	0.6	0.4	1	0.6	0.2

In contrast, SMILE proved to be CPU-intensive in almost all scenarios, often exceeding 90% processor usage. This monopolizes the mobile

CPU, making other applications unresponsive and increasing the risk of termination by the operating system. Moreover, SMILE's energy consumption was among the highest, frequently reaching peak values on the normalized scale. While it occasionally showed competitive runtime performance—for example in certain K-Means tasks—the excessive CPU and battery demands undermine its practicality in real-world mobile settings. Such characteristics restrict SMILE to high-end devices and exclude it from broader deployment.

Tribuo occupied an intermediate position: it could successfully complete certain tasks with acceptable efficiency, but its compatibility limitations and somewhat higher consumption compared to Weka reduce its attractiveness for large-scale or resource-constrained training.

Overall, the benchmarking results validate the thesis: efficient CPU usage and minimal energy demand are the key determinants of practical feasibility for on-device training. Weka's relatively low resource footprint confirms its suitability for mobile applications, while SMILE's heavy processor usage illustrates the limitations of libraries that fail to account for constrained environments. This underlines the importance of evaluating machine learning solutions not only by their algorithmic power but also by their efficiency in real-world, resource-limited contexts such as Android devices.

Integrating Ensemble Models into Microservice-Based Architectures

Thesis 3: The integration of ensemble machine learning models into microservice-based architectures demonstrates that heterogeneous models, developed in different programming languages and frameworks, can be unified through web services, thereby ensuring language independence and system interoperability.

The justification for this thesis lies in the architectural and practical mechanisms by which machine learning models are encapsulated and orchestrated within microservice environments. Machine learning models are typically developed in a variety of programming languages and supported by different libraries, which makes direct integration into larger software systems difficult. Wrapping these models as web services resolves this challenge by providing a standardized communication layer. Predictions and training requests are transmitted through RESTful interfaces using JSON, so the client system no longer depends on the internal representation of the model. This approach effectively eliminates language barriers: a classifier implemented in Java and a neural network implemented in Python can be accessed in the same way, ensuring uniformity at the system level.

Microservice architecture further reinforces this independence. Each model operates as a distinct service with its own data storage and computational logic, exposing only a public API to the rest of the system. This design guarantees loose coupling and allows the integration of heterogeneous components without imposing a common technology stack. It also facilitates scalability and maintainability, since new models can be introduced or existing ones updated without disrupting other parts of the application. Interoperability is thus achieved not only at the technical interface level but also at the architectural level, where services can be distributed, discovered, and managed independently while still contributing to a unified whole.

The use of ensemble learning provides direct evidence of effective unification. Ensemble techniques such as voting or bagging require consistent access to outputs from multiple models, and their successful application across services written in different languages demonstrates that these outputs can indeed be unified and combined. Whether performed on the client side by directly querying individual model

services, or on the server side through an aggregator microservice, ensemble construction illustrates how heterogeneous models can collaborate to produce a single predictive result.

Practical implementations confirm that this approach is feasible. Systems based on both direct communication with model services (Figure 3) and gateway-based aggregation (Figure 4) have been built and tested, showing that requests and responses can be handled reliably under load (Table 4). Although the gateway design introduces additional latency due to extra communication steps, both variants operate correctly and demonstrate that performance trade-offs can be managed according to application needs. The choice between these variants depends on factors such as the number of models used, the computational capacity of clients, and the distribution of network load, but in both cases the principle of language independence and interoperability remains intact.

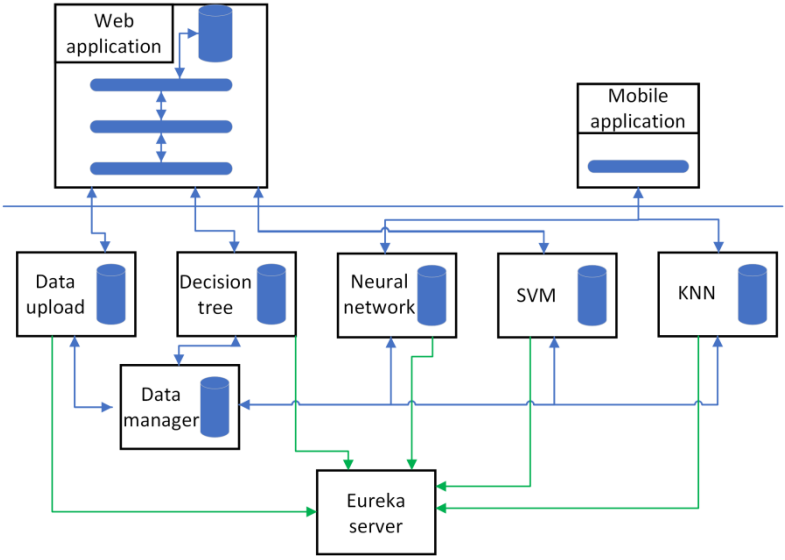


Figure 3. - Direct architecture variant with web and mobile clients

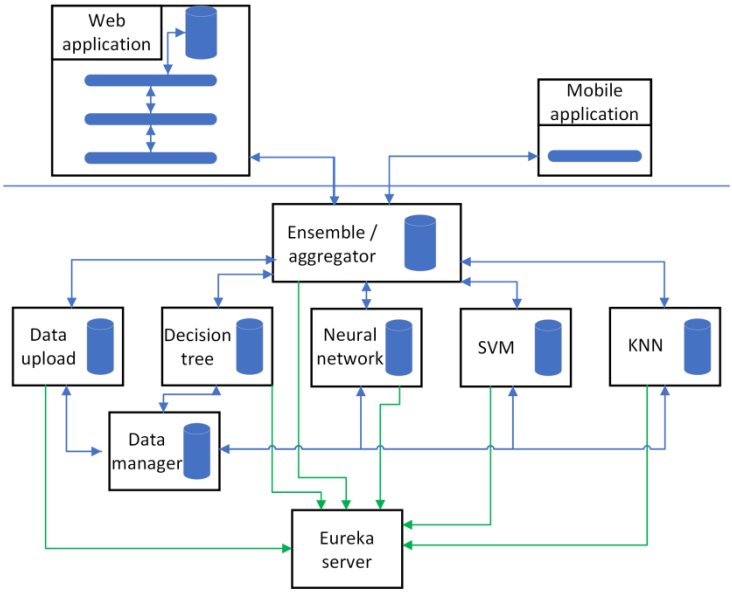


Figure 4. - Gateway architecture variant with web and mobile clients

Table 4. – Performance of Direct and Gateway Variants

Metric	Direct Variant	Gateway Variant
Number of virtual users	20	20
Number of requests served (per min)	1085	1004
Average requests per second (req/s)	13.71	12.96
Average response time (ms)	17	51
Minimum response time (ms)	9	18
Maximum response time (ms)	317	1221
90% of requests served within (ms)	24	121
Error rate (%)	0	0

In conclusion, the integration of ensemble machine learning models within microservice architectures shows that heterogeneity in

programming languages and frameworks does not constitute a barrier to unified operation. By abstracting models into web services and coordinating them through microservice principles, systems can achieve both flexibility and robustness. The ability to construct ensembles from diverse sources provides clear validation that language independence and system interoperability are not only achievable but practical, confirming the core claim of the thesis.

Numerosity Classification in the SMNIST Experiments

Thesis 4: In the SMNIST experiments, the performance of the deeplearning4j LeNet MNIST model, consistent with the other tested architectures, supports the finding that artificial neural networks, similar to humans, classify numerosities with higher accuracy when these remain below the OFS capacity of four.

The experiments presented in Section 5.1 under the label SMNIST for Machines provide support for the thesis that deep learning architectures originally developed for digit or image classification exhibit numerosity discrimination patterns similar to those observed in humans. The key observation is that artificial neural networks (ANNs) achieve high accuracy in classifying numerosities when the number of items is within the human Object File System (OFS) limit, but their performance declines as numerosity increases.

The justification rests first on the conceptual background. Human cognition relies on the OFS for accurate and rapid subitizing of up to four objects. Beyond this limit, estimation is guided by the Approximate Number System (ANS), which is less precise. The SMNIST for Humans experiments confirmed these findings, showing the expected accuracy drop beyond four. The novelty of the SMNIST for Machines studies lies in testing whether ANNs, without any explicit cognitive design, would display a similar performance trajectory.

Methodologically, the experiments were carefully designed. The SMNIST datasets mimic the structure of the MNIST benchmark but replace handwritten digits with images of up to nine dots (Figure 5). Several dataset variants (centered vs. non-centered dots, disjunct vs. overlapping placements, different pixel sizes) ensured that the task was non-trivial and that classification could not rely on simple memorization. Standard architectures—CNNs, MLPs, RNNs—were applied without tailoring them to subitizing, thus testing their spontaneous capacity to generalize to numerosity discrimination.

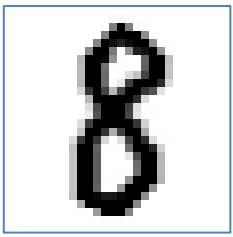
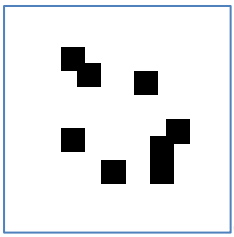

(a)

(b)

Figure 5. Two typical input images for MNIST (a) and SMNIST (b)

The results are consistent across architectures: performance is robust for small numerosities but systematically deteriorates as the number of dots increases. For instance, while CNNs and hierarchical RNNs reach near-perfect accuracy for numerosities up to four, their accuracy decreases markedly with seven, eight or nine objects (Table 5). This mirrors the human OFS limitation and suggests that neural networks encode small numerosities in a manner comparable to subitizing. Importantly, the decline is not random but follows the same boundary observed in cognitive psychology, strengthening the analogy between human and machine performance.

Table 5.- „SMNIST for Machines" 2. series measurements

Program	4 object	5 object	6 object	7 object	8 object

Tensorflow 0.9.0, mnist_softmax.py	0.6317	0.3188	0.2334	0.1936	0.1658
Keras 2.2.4, mnist_cnn.py	0.9099	0.7055	0.7599	0.7285	0.6568
Keras/Hierarchical RNN	0.9993	0.9996	0.9442	0.9996	0.9993
PyTorch, cifar10_tutorial.py	0.8758	0.8589	0.723	0.556	0.6733
deeplearning4j LeNet MNIST	0.7743	0.5329	0.4770	0.3671	0.2977
Swift TF MNIST	0.6432	0.4906	0.3102	0.2896	0.1819
Swift TF CIFAR PyTorch	0.6729	0.6218	0.4796	0.4156	0.4802

The implication is that numerosity discrimination in ANNs is an emergent property of general visual classification mechanisms. Just as humans have not evolved a symbolic arithmetic system for small numerosities but rely on OFS, so too do ANNs display a natural facility for small sets, with limitations beyond that threshold. The convergence between the two systems—biological and artificial—indicates that the structural constraints of pattern recognition may inherently give rise to subitizing-like behavior.

The LeNet model, in particular, illustrates this trend clearly: in the Series 2 datasets, it achieved an accuracy of 0.774 for four-dot images, yet its performance dropped to 0.532 for five dots, 0.477 for six, 0.367 for seven, and only 0.297 for eight. In the “SMNIST for Anyone” series, where dots were replaced by simple symbols, the same pattern

was observed: relatively high accuracy at four objects, followed by a sharp decline across larger numerosities.

References

[1] K. Kourou, T. P. Exarchos, K. P. Exarchos, M. V. Karamouzis, and D. I. Fotiadis, “Machine learning applications in cancer prognosis and prediction,” Computational and Structural Biotechnology Journal, vol. 13, pp. 8–17, Nov. 2014, doi: 10.1016/j.csbj.2014.11.005.

[2] M. Kukar, I. Kononenko, C. Grošelj, K. Kralj, and J. Fettich, “Analysing and improving the diagnosis of ischaemic heart disease with machine learning,” Artificial Intelligence in Medicine, vol. 16, no. 1, pp. 25–50, May 1999, doi: 10.1016/s0933-3657(98)00063-3.

[3] S. Li et al., “Hippocampal shape analysis of Alzheimer disease based on machine learning methods,” American Journal of Neuroradiology, vol. 28, no. 7, pp. 1339–1345, Aug. 2007, doi: 10.3174/ajnr.a0620.

[4] M. Chen, Y. Hao, K. Hwang, L. Wang, and L. Wang, “Disease prediction by machine learning over big data from healthcare communities,” IEEE Access, vol. 5, pp. 8869–8879, Jan. 2017, doi: 10.1109/access.2017.2694446.

[5] M. S. Bartlett, G. Littlewort, M. Frank, C. Lainscsek, I. Fasel, and J. Movellan, “Recognizing Facial Expression: Machine Learning and Application to Spontaneous Behavior,” 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05), vol. 2. IEEE, pp. 568–573. doi: 10.1109/cvpr.2005.297.

[6] S. Zander, T. Nguyen, and G. Armitage, “Automated traffic classification and application identification using machine learning,” The IEEE Conference on Local Computer Networks 30th Anniversary (LCN’05)l. IEEE, pp. 250–257, 2005. doi: 10.1109/lcn.2005.35.

[7] K. Alsabti, S. Ranka, and V. Singh, “CLOUDS: a decision tree classifier for large datasets,” Knowledge Discovery and Data Mining, pp. 2–8, Aug. 1998, [Online]. Available: https://aaaipress.org/Papers/KDD/1998/KDD98-001.pdf

[8] S.-M. Lee and P. A. Abbott, “Bayesian networks for knowledge discovery in large datasets: basics for nurse researchers,” Journal of Biomedical Informatics, vol. 36, no. 4–5, pp. 389–399, Aug. 2003, doi: 10.1016/j.jbi.2003.09.022.

[9] H. Yu, J. Yang, and J. Han, “Classifying large data sets using SVMs with hierarchical clusters,” Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, pp. 306–315, Aug. 24, 2003. doi: 10.1145/956750.956786.

[10] “PassMark Android Benchmark Charts - CPU Rating.” https://www.androidbenchmark.net/cpumark_chart.html, (last visited 24 August 2025).

[11] “Smartphone Processors - Benchmark List,” Notebookcheck, May 20, 2019. https://www.notebookcheck.net/Smartphone-Processors-Benchmark-List.149513.0.html, (last visited 24 August 2025).

[12] Statista, “Share of global mobile website traffic 2015-2024,” Statista, Jan. 28, 2025. https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobiledevices/, (last visited 24 August 2025).

[13] Statista, “U.S. daily media usage time via mobile 2019-2023,” Statista, Jul. 08, 2025. https://www.statista.com/statistics/469983/time-spent-mobile-media-type-usa, (last visited 24 August 2025).

[14] “Mobile vs. Desktop Usage in 2020,” Perficient. https://www.perficient.com/insights/research-hub/mobile-vs-desktop-usage, (last visited 24 August 2025).

[15] G. Seni and J. F. Elder, “Ensemble methods in data mining: Improving accuracy through combining predictions,” Synthesis Lectures on Data Mining and Knowledge Discovery, vol. 2, no. 1, pp. 1–126, Jan. 2010, doi: 10.2200/s00240ed1v01y200912dmk002.

[16] A. Guazzelli, W.-C. Lin, and T. Jena, PMML in action: unleashing the power of open standards for data mining and predictive analytics. Seattle: CreateSpace, 2010.

[17] W.-F. Lin et al., “ONNC: A Compilation Framework Connecting ONNX to Proprietary Deep Learning Accelerators,” 2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS). IEEE, pp. 214–218, Mar. 2019. doi: 10.1109/aicas.2019.8771510.

[18] Onnx, “GitHub - onnx/onnx: Open standard for machine learning interoperability,” GitHub. https://github.com/onnx/onnx, (last visited 24 August 2025).

[19] B. Amos, H. Turner, and J. White, “Applying machine learning classifiers to dynamic Android malware detection at scale,” 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC). IEEE, pp. 1666–1671, July 2013. doi: 10.1109/iwcmc.2013.6583806.

[20] N. Peiravian and X. Zhu, “Machine Learning for Android Malware Detection Using Permission and API Calls,” 2013 IEEE 25th International Conference on Tools with Artificial Intelligence. IEEE, pp. 300–305, Nov. 2013. doi: 10.1109/ictai.2013.53.

[21] J. Sahs and L. Khan, “A Machine Learning Approach to Android Malware Detection,” 2012 European Intelligence and Security Informatics Conference. IEEE, pp. 141–147, Aug. 2012. doi: 10.1109/eisic.2012.34.

[22] Ž. Jovanović, D. Jagodić, D. Vujičić, and S. Ranđić, “Java Spring Boot REST Web Service Integration with Java Artificial Intelligence Weka Framework,” in Proc. International Scientific Conference UNITEH 2017, Gabrovo, Bulgaria, Nov. 17–18, 2017, pp. 270–274. [Online]. Available: https://www.unitechsp.tugab.bg/images/papers/2017/s5/s5_p238.pdf

[23] M. Li et al., “Scaling Distributed Machine Learning with the Parameter Server,” in Proc. 11th USENIX Symposium on Operating Systems Design and Implementation. (OSDI ’14), Broomfield, CO, USA, Oct. 6–8, 2014, pp. 583–598.

[24] T. Kraska, A. Talwalkar, J. C. Duchi, R. Griffith, M. J. Franklin, and M. I. Jordan, “MLbase: A Distributed Machine-learning System,” in Proc. 6th

Biennial Conference on Innovative Data Systems Research (CIDR ’13), Asilomar, CA, USA, Jan. 2013.

[25] E. R. Sparks et al., “MLI: An API for Distributed Machine Learning,” 2013 IEEE 13th International Conference on Data Mining. IEEE, pp. 1187–1192, Dec. 2013. doi: 10.1109/icdm.2013.158.

[26] Meng, X., et al. "MLlib: machine learning in apache spark," in The Journal of Machine Learning Research, vol. 17, no. 1, pp. 1235–1241, 2016.

[27] A. McIntosh, S. Hassan, and A. Hindle, “What can Android mobile app developers do about the energy consumption of machine learning?,” Empirical Software Engineering, vol. 24, no. 2, pp. 562–601, June 2018, doi: 10.1007/s10664-018-9629-2.

[28] A. Ignatov et al., “AI Benchmark: Running Deep Neural Networks on Android Smartphones,” Lecture Notes in Computer Science. Springer International Publishing, pp. 288–314, 2019. doi: 10.1007/978-3-030-11021-5_19.

[29] A. Kulkarni, N. Mhalgi, S. Gurnani, and N. Giri, “Pothole Detection System using Machine Learning on Android,” International Journal of Emerging Technology and Advanced Engineering, vol. 4, no. 7, pp. 360–364, Jul. 2014. [Online]. Available: http://www.ijetae.com/files/Volume4Issue7/IJETAE_0714_55.pdf

[30] R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore, “PMLB: a large benchmark suite for machine learning evaluation and comparison,” BioData Mining, vol. 10, no. 1, Dec. 2017, doi: 10.1186/s13040-017-0154-4.

[31] C. Coleman et al., “Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark,” ACM SIGOPS Operating Systems Review, vol. 53, no. 1, pp. 14–25, July 2019, doi: 10.1145/3352020.3352024.

[32] P. Mattson, C. Cheng, C. Coleman, G. Diamos, P. Micikevicius, D. Patterson, et al., “MLPerf Training Benchmark,” in Proceedings of Machine Learning and Systems, vol. 2, I. Dhillon, D. Papailiopoulos, and V. Sze, Eds.,

2020, pp. 336–349. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2020/file/411e39b117e885341f25efb8912945f7-Paper.pdf

[33] F. Pirotti, F. Sunar, and M. Piragnolo, “BENCHMARK OF MACHINE LEARNING METHODS FOR CLASSIFICATION OF A SENTINEL-2 IMAGE,” International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences, vol. XLI-B7, pp. 335–340, June 2016, doi: 10.5194/isprsarchives-xli-b7-335-2016.

[34] E. Frank, M. A. Hall, and I. H. Witten, “The WEKA Workbench: Online Appendix for Data Mining: Practical Machine Learning Tools and Techniques,” Morgan Kaufmann, 4th ed., Online appendix, 2016. [Online]. Available:
https://ml.cms.waikato.ac.nz/weka/Witten_et_al_2016_appendix.pdf.

[35] H. Li, “Smile - home.” https://haifengl.github.io/. (Last visited 24 August 2025)

[36] Oracle, Tribuo: Machine Learning in Java. https://tribuo.org/ (Last visited 24 August 2025)

[37] T. Chen and C. Guestrin, “XGBoost,” Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, pp. 785–794, Aug. 13, 2016. doi: 10.1145/2939672.2939785.

[38] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, et al., “TensorFlow: A System for Large-Scale Machine Learning,” in Proceedings of USENIX Symposium on Operating Systems Design and Implementation. (OSDI ’16), Savannah, GA, USA, Nov. 2–4, 2016, pp. 265–283.

[39] Deeplearning4j Development Team, “Deeplearning4j: Open-source Distributed Deep Learning for the JVM,” 2016. GitHub. http://deeplearning4j.org/. (Last visited 24 August 2025)

[40] H2oai, GitHub - h2oai/h2o-3. https://github.com/h2oai/h2o-3

[41] P. K. Falk, “Tech Services on the Web: MALLET-MAchine Learning for LanguagE Toolkit; http://mallet.cs.umass.edu/,” Technical Services

Quarterly, vol. 31, no. 4, pp. 410–411, Sept. 2014, doi: 10.1080/07317131.2014.943038.

[42] M.-O. Pahl and M. Loipfinger, “Machine learning as a reusable microservice,” NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium. IEEE, pp. 1–7, Apr. 2018. doi: 10.1109/noms.2018.8406165.

[43] Y.-D. Bromberg and L. Gitzinger, “DroidAutoML: A Microservice Architecture to Automate the Evaluation of Android Machine Learning Detection Systems,” Lecture Notes in Computer Science. Springer International Publishing, pp. 148–165, 2020. doi: 10.1007/978-3-030-50323-9_10.

[44] J. L. Ribeiro, M. Figueredo, A. Araujo, N. Cacho, and F. Lopes, “A Microservice Based Architecture Topology for Machine Learning Deployment,” 2019 IEEE International Smart Cities Conference (ISC2). IEEE, pp. 426–431, Oct. 2019. doi: 10.1109/isc246665.2019.9071708.

[45] D. C. Attota, V. Mothukuri, R. M. Parizi, and S. Pouriyeh, “An Ensemble Multi-View Federated Learning Intrusion Detection for IoT,” IEEE Access, vol. 9, pp. 117734–117745, 2021, doi: 10.1109/access.2021.3107337.

[46] P. Salza, E. Hemberg, F. Ferrucci, and U.-M. O’Reilly, “cCube,” Proceedings of the Genetic and Evolutionary Computation Conference Companion. ACM, pp. 137–138, July 15, 2017. doi: 10.1145/3067695.3076089.

[47] J. Gruendner et al., “KETOS: Clinical decision support and machine learning as a service – A training and deployment platform based on Docker, OMOP-CDM, and FHIR Web Services,” PLoS ONE, vol. 14, no. 10, p. e0223010, Oct. 2019, doi: 10.1371/journal.pone.0223010.

[48] M. Chippa, A. Priyadarshini, and R. Mohanty, “Application of Machine Learning Techniques to Classify Web Services,” 2019 IEEE International Conference on Intelligent Techniques in Control, Optimization and Signal Processing (INCOS). IEEE, pp. 1–7, Apr. 2019. doi: 10.1109/incos45849.2019.8951339.

[49] H. Alipour and Y. Liu, “Online machine learning for cloud resource provisioning of microservice backend systems,” 2017 IEEE International Conference on Big Data (Big Data). IEEE, pp. 2433–2441, Dec. 2017. doi: 10.1109/bigdata.2017.8258201.

[50] H. Chang, M. Kodialam, T. V. Lakshman, and S. Mukherjee, “Microservice Fingerprinting and Classification using Machine Learning,” 2019 IEEE 27th International Conference on Network Protocols (ICNP). IEEE, pp. 1–11, Oct. 2019. doi: 10.1109/icnp.2019.8888077.

[51] H. Harms, C. Rogowski, and L. Lo Iacono, “Guidelines for adopting frontend architectures and patterns in microservices-based systems,” Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. ACM, pp. 902–907, Aug. 21, 2017. doi: 10.1145/3106237.3117775.

[52] D. Taibi, V. Lenarduzzi, C. Pahl, and A. Janes, “Microservices in agile software development,” Proceedings of the XP2017 Scientific Workshops. ACM, pp. 1–5, May 22, 2017. doi: 10.1145/3120459.3120483.

[53] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, “A Survey on Distributed Machine Learning,” ACM Computing Surveys, vol. 53, no. 2, pp. 1–33, Mar. 2020, doi: 10.1145/3377454.

[54] J. A. Valdivia, A. Lora-González, X. Limón, K. Cortes-Verdin, and J. O. Ocharán-Hernández, “Patterns Related to Microservice Architecture: a Multivocal Literature Review,” Programming and Computer Software, vol. 46, no. 8, pp. 594–608, Dec. 2020, doi: 10.1134/s0361768820080253.

[55] H. Chawla and H. Kathuria, “Implementing Microservices,” Building Microservices Applications on Microsoft Azure. Apress, pp. 21–41, 2019. doi: 10.1007/978-1-4842-4828-7_2.

[56] K. S. Prasad Reddy, Beginning Spring Boot 2. Apress, 2017. doi: 10.1007/978-1-4842-2931-6.

[57] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, et al., “Scikit-learn: Machine Learning in Python,” Journal of Machine

Learning Research, vol. 12, pp. 2825–2830, Oct. 2011. doi: 10.48550/arXiv.1201.0490.

[58] M. Grinberg, Flask web development: developing web applications with Python, 1. ed. Beijing Köln: O'Reilly, 2014.

List of relevant publications

Referred journal papers

[J1] N. Bátfai et al., “Object file system software experiments about the notion of number in humans and machines,” Cognition Brain Behavior an Interdisciplinary Journal, vol. 23, no. 4, pp. 257–280, Dec. 2019, doi: 10.24193/cbb.2019.23.15.

[J2] M. Szabó, “Distributed Machine Learning Using Data Parallelism on Mobile Platform,” Journal of Mobile Multimedia, vol. 16, no. 3, pp. 317-334, Sept. 2020, doi: 10.13052/jmm1550-4646.1633.

[J3] M. Szabó, “Building Ensemble Models with Web Services on Microservice Architecture,” Informatica, vol. 48, no. 7, pp. 1-10, Mar. 2024, doi: 10.31449/inf.v48i7.4918.

Papers in conference proceedings

[I1] M. Szabó, ”Machine Learning on Android with Oracle Tribuo, SMILE and Weka,” CEUR Workshop Proceedings, vol. 2874, pp. 176-186, 2021

[I2] N. Bátfai, T. Tutor, Z. Bartha, A. Czanik, M. Szabó, ”Red Flower Hell: a Minecraft MALMÖ Challenge to Support Introductory Programming Courses,” CEUR Workshop Proceedings, vol. 2874, pp. 56-66, 2021

UNIVERSITY AND NATIONAL LIBRARY
UNIVERSITY OF DEBRECEN
H-4002 Egyetem tér 1, Debrecen
Phone: +3652/410-443, email: publikaciok@lib.unideb.hu

Registry number: DEENK/494/2025.PL
Subject: PhD Publication List

Candidate: Máté Szabó
Doctoral School: Doctoral School of Informatics
MTMT ID: 10071945

List of publications related to the dissertation

Foreign language scientific articles in international journals (3)

1. **Szabó, M.**: Building Ensemble Models with Web Services on Microservice Architecture. *Informatica (Slovenia). 48* (7), 1-10, 2024. ISSN: 0350-5596. DOI: http://dx.doi.org/10.31449/inf.v48i7.4918

2. **Szabó, M.**: Distributed Machine Learning Using Data Parallelism on Mobile Platform. *JMM. 16* (3), 317-334, 2020. ISSN: 1550-4646. DOI: http://dx.doi.org/10.13052/jmm1550-4646.1633

3. Bátfai, N., Papp, D., Bogacsovics, G., **Szabó, M.**, Simkó, V. S., Bersenszki, M., Szabó, G., Kovács, L., Kovács, F., Varga, E. S.: Object file system software experiments about the notion of number in humans and machines. *Cognition, Brain, Behavior. 23* (4), 257-280, 2019. ISSN: 2247-9228. DOI: http://dx.doi.org/10.24193/cbb.2019.23.15

Foreign language conference proceedings (2)

4. **Szabó, M.**: Machine Learning on Android with Oracle Tribuo, SMILE and Weka. In: Proceedings of the 1st Conference on Information Technology and Data Science. Ed.: István Fazekas, András Hajdu, Tibor Tómács, CEUR Workshop Proceedings, Debrecen, 176-186, 2021, (CEUR Workshop Proceedings, ISSN 1613-0073 ; 2874.)

5. Bátfai, N., Tutor, T., Bartha, Z., Czanik, A., **Szabó, M.**: Red Flower Hell: a Minecraft MALMÖ Challenge to Support Introductory Programming Courses. In: Proceedings of the 1st Conference on Information Technology and Data Science. Ed.: István Fazekas, András Hajdu, Tibor Tómács, CEUR Workshop Proceedings, Debrecen, 56-66, 2021, (CEUR Workshop Proceedings, ISSN 1613-0073 ; 2874.)

UNIVERSITY of DEBRECEN

UNIVERSITY AND NATIONAL LIBRARY
UNIVERSITY OF DEBRECEN
H-4002 Egyetem tér 1, Debrecen
Phone: +3652/410-443, email: publikaciok@lib.unideb.hu

List of other publications

Foreign language scientific articles in Hungarian journals (1)

6. Bátfai, N., Besenczi, R., Jeszenszky, P., **Szabó, M.**, Ispány, M.: Markov modeling of traffic flow in Smart Cities.
   Ann. Math. Inform. 53, 21-44, 2021. ISSN: 1787-5021.
   DOI: http://dx.doi.org/10.33039/ami.2021.04.008

Foreign language conference proceedings (2)

7. Jeszenszky, P., Besenczi, R., **Szabó, M.**, Ispány, M.: Estimating road traffic flows in macroscopic Markov model.
   In: 2022 IEEE 2nd Conference on Information Technology and Data Science (CITDS), IEEE, Piscataway, 136-141, 2022. ISBN: 9781665496537
8. Bátfai, N., **Szabó, M.**: Possible Neural Models to Support the Design of Prime Convo Assistant.
   In: Proceedings of the 1st Conference on Information Technology and Data Science. Ed.: István Fazekas, András Hajdu, Tibor Tómács, CEUR Workshop Proceedings, Debrecen, 46-55, 2021, (CEUR Workshop Proceedings, ISSN 1613-0073 ; 2874.)

The Candidate's publication data submitted to the Tudóstér have been validated by DEENK on the basis of the Journal Citation Report (Impact Factor) database.

27 August, 2025

DEBRECENI EGYETEM * Egyetemi és Nemzeti Könyvtár *