Design and performance analysis of applications using machine learning

Egyetemi doktori (PhD) értekezés

Szabó Máté
Témavezető: Dr. Ispány Márton

DEBRECENI EGYETEM
Természettudományi és Műszaki Tudományi Doktori Tanács
Informatikai Tudományok Doktori Iskola
Debrecen, 2025

Ezen értekezést a Debreceni Egyetem Természettudományi és Műszaki Tudományi Doktori Tanács, Informatikai Tudományok Doktori Iskola Alkalmazott információ technológia és elméleti háttere programja keretében készítettem a Debreceni Egyetem műszaki doktori (PhD) fokozatának elnyerése céljából.
Nyilatkozom arról, hogy a tézisekben leírt eredmények nem képezik más PhD disszertáció részét.
Debrecen, 2025. . . . . .

……
a jelölt aláírása

Tanúsítom, hogy Szabó Máté doktorjelölt 2018- 2025 . . között a fent megnevezett Doktori Iskola Alkalmazott információ technológia és elméleti háttere programjának keretében irányításommal végezte munkáját. Az értekezésben foglalt eredményekhez a jelölt önálló alkotó tevékenységével meghatározóan hozzájárult. Nyilatkozom továbbá arról, hogy a tézisekben leírt eredmények nem képezik más PhD disszertáció részét.
Az értekezés elfogadását javasolom.

Debrecen, 2025.. . . . . …….
a témavezető aláírása

Design and performance analysis of applications using machine learning

Értekezés a doktori (Ph.D.) fokozat megszerzése érdekében
a műszaki tudományterület, informatikai tudományágban

Írta: Szabó Máté okleveles programtervező informatikus

Készült a Debreceni Egyetem Informatikai Tudományok doktori iskolája
Alkalmazott információ technológia és elméleti háttere programja) keretében

Témavezető: Dr. Ispány Márton

Az értekezés bírálói:

Dr. .....
Dr. .....

A bírálóbizottság:

elnök: Dr. .....
tagok: Dr. .....
Dr. .....
Dr. .....
Dr. .....

Az értekezés védésének időpontja: 20… . …… … .

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data is information that is gathered by measurements, analysis, research, or observations. Because of its diversity, data can be anything and can come from anywhere, and as the whole world is full of it, it must be useful for everyone. People make decisions and organize their lives based on the data that was around them in their lifetime. Just like computers, we store this information in our brains to process it to make the rules and boundaries of our own system. This process of data collection and processing begins in the womb during the prenatal phase of life. Some of the first kind of data can be the voice of the mother as the only constant thing that is present, or the touch of the surrounding organs, or the sound of blood flowing, or heartbeats. The experience of these means the safety for babies, as these are the only kind of data that they ever collected and processed. After birth, the concept of safety is still connected to these data, so whenever they feel uneasy about a situation, the mother's voice can calm them. As they get older, they encounter a variety of data. For example, some of them will define their environment, such as dimensions, shapes, and colors, and some of them are used to express thoughts, such as body language or later words that are connected to concepts. As time progresses, their mind can process more complex data that is often derived from the ones that they experienced earlier. For example, the knowledge of shapes is useful when learning the letters and numbers, or as the number of known words expands, sentences will be formed to express thoughts more precisely, like the word hungry could transform into hungry for apples. We can clearly see that a person's development is about data, and as they get older, with more complex data, they can achieve more.

When we think about the history of computer science, we can easily draw parallels between the human brain and computer science development in terms of data storage and processing. Some of these milestones can be mentioned, like drum memory, cassette tapes, floppy discs, hard disks, optical disks, or flash memory drives. With these, computers could manage more and more data with every evolutionary step, and thus, they could process this increasing amount of data to achieve more complex goals. For example computer should be able to show us a visual image, but based on the maturity of these systems, the early ones could only render some pixels in black and white, but as time progresses and they had more data, these pixels had an RGB color and later, the size of the images could grow to tell more information to the viewer like the evolution of

drawing from stick figures to exact human faces and bodies. As computers can manage the diversity and volume of data, they can be used to help or sometimes replace the human workforce. The data that was previously stored on paper moved to drives, and thus, the decisions made by humans based on these papers can now be made by computers. As previously mentioned, personal development and computer science evolution are similar to each other, but ours is a much slower process. The challenge is that both of them often do the same kind of tasks, like calculation, decision making, painting, or anything else, but we do it with much less data or experience than computers that could learn a lifetime of experience in hours.

If we go further with the conclusions of the previous philosophizing parts, that data is the essence of everything and defines everything, we could state that the 21st century's most valuable resource is data. Because of this, almost every company tries to gather as much of it as they can, and if they have some, they want to get data to work. For example, shops know people's shopping habits, and they want to know what you will buy. Warehouse management knows what items they have, but they want to know how much they should order to satisfy customer needs. Some are interested in the weather, and some are interested in stocks. Each of the previous sentences is achievable with data collection, processing, and pattern construction. These goals give companies one step ahead of the competition, and from that, they could get a business advantage.

Of course, computer systems do not know these patterns or conclusions on their own; they need to learn them too from the provided data, and for that, they need the tools of machine learning (ML), which is a central topic of this thesis. ML has a collection of algorithms that can be used for different purposes, like prediction, classification, and clustering. Based on the goal, models can be separated into three categories: supervised learning, unsupervised learning, and semi-supervised learning. Supervised needs labeled datasets to train the model that can classify data or predict outcomes. It works like we have observations on a specific area, and we know what the outcome was based on the attributes we chose, and from these, the algorithm makes patterns, which can be applied to new data, whose outcome is unknown, to predict it. Some supervised methods are neural network, decision tree, random forest, support vector machine, linear regression, and logistic regression. In unsupervised learning, there are no labels, just the observations. Here, we do not know what their true class or value is; we can only start from the difference between the data. It is like we are in a field we have no expertise in, and the only thing we can state is that our two observations are not the same or not even close to each other. Some unsupervised methods are k-means clustering, hierarchical clustering, and DBSCAN. Semi-supervised

learning is a state between the previous two, where we have labeled data, but not all data is labeled.

Besides machine learning, the web applications are also the focus of this thesis. Application development's target platforms have rearranged over the past years, and while earlier desktop applications were the most popular, later mobile applications have overtaken the top position as smartphones have begun to spread worldwide. Nowadays, this mobile oriented attitude has changed, as these devices seem to have reached their maximum potential, and because of this, web applications and recently AI applications are on top. There are many reasons why the transition from desktop to web seems like a long-term trend. One of them can be their accessibility, as there is no installation needed for end users, so companies can introduce a new software in minutes without installing it on the computers of the whole company. Their advantage is cross-platform compatibility, too, as they can be used by any kind of device and operating system that can send and handle an HTTP request and response. They provide the ease of updates and maintenance as they are hosted on servers; only one application needs to be updated and maintained, and all users will use the same software version. If the architecture is well designed, scalability is easily achievable too, with the adjustment of server resources. Securing web applications also differs from desktop applications, as web applications are easier to access, which means developers should pay extra attention to vulnerabilities. An important advantage is the integration with other services, as they can easily integrate with other online services through APIs. This feature will play a bigger role in later chapters, as this thesis focuses on machine learning, web applications, and mobile applications and their integrability.

The third development area is related to mobile devices and applications regarding their machine learning capability. Although native mobile application development shows a somewhat decreasing trend, it is still a very popular area. As smartphones became more accessible and affordable, they became the primary computing device for many people. The hardware of these evolved so fast compared to the old phone generations that it induced the development of applications whose functionality seemed impossible earlier. These devices have more sensors each year, more powerful processors, and high-quality cameras that can produce and process a bigger variety of data. The always-connected nature can be an advantage and a disadvantage, too, as it makes mobile applications the most accessible form of software, and their users can perform tasks, consume content, and stay connected from anywhere at any time, which can lead to dependence and can shift the boundaries between real and online life. Push notifications allow apps to engage with users in real time, providing updates, reminders, and alerts that keep users coming back to the app. These applications

provide personalized experiences to users as they can use data from various sensors and user inputs. Some of them can be location-based services, personalized content feeds, or notifications. The platform's popularity greatly increased with the appearance of social media, instant messaging, mobile gaming, streaming services, and e-commerce platforms. The app stores and their related business models were enticing to the developers, as they created a robust ecosystem to monetize the applications through in-app purchases, direct sales, or advertising.

This thesis connects the three loosely related topics: machine learning, mobile applications, and web applications. Each of them is very popular individually, but when they are applied together, powerful and efficient applications can be developed. The thesis presents three applications and their evaluation to present the machine learning capabilites of mobile devices with parallelization where reasonable, and then the integration of these and standard machine learning into existing web and microservice based systems, leading to the conclusion that ml models can be applied to almost any kind of computer software that has relations to web.

The second chapter investigates the potential of leveraging data parallelism for distributed machine learning on mobile devices. It addresses the challenges associated with processing large-scale datasets, which often exceed the capabilities of traditional machine learning approaches due to hardware constraints. The research proposes a client-server architecture that enables the distribution of machine learning model training across a cluster of mobile devices. This system comprises a web service that oversees dataset management and model persistence, along with a mobile client responsible for the training and evaluation of models. The web service facilitates critical operations such as data slicing, model storage, and the creation of ensemble models, ensuring an efficient distributed training process. The chapter demonstrates that it is not only feasible but also efficient to perform distributed training on mobile platforms, utilizing the computing power of these devices that are often underutilized. The approach detailed in the chapter highlights the potential for mobile devices to handle complex machine learning tasks, which traditionally require more robust computing resources. The study finds that the performance of distributed training on mobile devices is contingent on the specific implementation of the machine learning algorithms employed.

The third chapter focuses on evaluating the feasibility and performance of using three Java-based machine learning libraries - Oracle Tribuo, SMILE, and Weka - on Android devices. Given that Python is the most popular language for machine learning, it has limitations when it comes to compatibility with Android. As a result, the study explores alternatives using Java and Kotlin, which are more

native to Android development. It details the process of porting these libraries to Android, highlighting the challenges encountered due to incompatibilities between Android and Java code. Despite these challenges, the libraries were successfully ported, and the chapter then assesses their performance by developing an Android application that trains machine learning models (Random Forest, Support-Vector Machine, and K-means) using these libraries. The study compares the runtime, memory usage, CPU usage, and battery consumption across the three libraries on different datasets, revealing that Weka performs best in terms of resource efficiency, especially with larger datasets and more complex models. SMILE and Tribuo are also evaluated, showing their strengths in certain scenarios, but Weka is recommended for its overall balance of efficiency and capability.

The fourth chapter explores the integration of machine learning models with web services within a microservice architecture. The focus is on developing ensemble models that act as wrapper models, allowing for the combination of pre-trained models written in different programming languages and environments, such as Scikit Learn, TensorFlow, Weka, and Deeplearning4j. The study proposes two architectural variants - direct and gateway - for integrating these models into web applications. The study highlights the advantages and challenges of using microservices for machine learning, emphasizing the flexibility of using different languages and tools while addressing the potential communication overhead in such systems.

The first part of the fifth chapter presents a series of software experiments aimed at understanding how humans and machines perceive and classify numerosity (the number of items in a set). The study explores two types of experiments: "SMNIST for Humans" and "SMNIST for Machines." The former investigates how the Object File System (OFS) in humans allows quick and accurate recognition of small quantities (up to four objects) without counting, aligning with cognitive psychology findings. The latter examines how well deep learning models, originally designed for other tasks, can learn to distinguish and classify numerosities in images. The research aims to compare these machine learning results with human performance, particularly in scenarios where the number of objects exceeds the typical capacity of human OFS. The chapter also discusses broader implications for understanding the cognitive evolution of numerical perception in both biological and artificial systems.

The second part of the fifth chapter presents an approach to teaching programming through a game-based challenge using Minecraft's MALMÖ mod. Developed at the University of Debrecen, the challenge involves programming agents to navigate a Minecraft environment, collecting red flowers while avoiding dangers. The tasks increase in complexity, with the possibility of

integrating AI concepts like heuristic algorithms and Q-learning. This approach aims to enhance student engagement and improve programming skills, demonstrating the effectiveness of using interactive, game-based learning in educational settings.

Chapter 2

Machine Learning on Mobile Devices: A Data-Parallel Distributed Training Architecture

This chapter is based on my first own publication, entitled Distributed machine learning using data parallelism on mobile platform [O1].

Machine learning is a very popular area in informatics, because most of the applications and services use it directly or indirectly. The application of machine learning techniques can be found in nearly every discipline; for example, they are used in healthcare, cars, politics, and commerce. In healthcare, it can be used to predict diseases or medical parameters based on the collected data. To be more concrete, researchers use machine learning to predict cancer [1], to improve the diagnosis of ischemic heart disease [2], for hippocampal shape analysis [3], and for general disease prediction [4]. In other fields, there are many special applications, like facial expression recognition [5] or automated traffic classification [6]. There is a need for the evolution of machine learning techniques because the size of the datasets is growing continuously year by year. Traditional machine learning has its limits, especially in handling big data, because the complexity and size of datasets grow faster than the hardware's computing capacity. Of course, there are several methods for handling large datasets with traditional machine learning techniques, like the clouds decision tree [7], Bayesian networks [8], and support vector machines [9]. In addition to data volume, another significant challenge in machine learning is the high dimensionality of datasets, where the number of attributes can be extremely large. This so-called "curse of dimensionality" can lead to increased computational cost and overfitting, making model training more difficult. Dimensionality reduction techniques such as Principal Component Analysis (PCA) or feature selection methods can help mitigate these issues. Moreover, as the number of records grows, scalability becomes critical. Distributed computing frameworks and parallelism strategies—like data parallelism or model parallelism—are essential to efficiently process large-scale datasets across multiple nodes or devices.

Mobile phones have become the most popular devices, and because of this popularity, their hardware has evolved quite fast. Each year, the computing capacity of the flagship mobiles grows significantly [10, 11], and most of them are even capable of using machine learning models. There is no doubt that the popularity of these devices will grow more in 10 years [12–14]. The problem

with the increasing number of mobile devices is that most of the time, their processors and GPUs are idle, so their users use only a fragment of the computing capacity.

This chapter focuses on the utilization of mobile processors for machine learning purposes, especially with large datasets. My aim was to create an architecture that is capable of processing machine learning models trained on phones. There is a reference implementation of this architecture, so the performance and other properties are measured. This chapter deals with challenges from multiple areas, such as transferring models between devices, verifying the received models and reconstructing them, and training neural networks on mobiles. One of the most important parts of this system is the web service, which can handle incoming machine learning models, persist them in the database, construct ensemble models, and slice datasets. The other important part is the mobile application, which can train simple neural networks, use trained models, and communicate with the web service. Constructing ensemble models is an advantage of this system because with them, the accuracy can be increased [15]. For handling large datasets, this system uses data parallelism, so each device can train a model on a part of the data. For the challenge of transferring these models to the server, I used byte array serialization, because in this form, the received model can be verified and then deserialized into the same model.

2.1. Related work

This chapter can be placed in many topics, like distributed machine learning, mobile machine learning, and machine learning model exchange formats.

There are many model exchange formats, and each of them approaches the problem differently. Predictive Model Markup Language (PMML) is an XML format for predictive models [16]. In this language, the model is defined by a header, a data dictionary, data transformations, the model itself, the mining schema, the targets, and the outputs. It supports multiple model types, like neural networks, Gaussian processes, Bayesian networks, and support vector machines. ONNX is another project to share machine learning models between frameworks. It defines a computation graph model, built-in operators, and standard data types. The supported frameworks are Caffe2, Chainer, Cognitive Toolkit, MXNet, PyTorch, PaddlePaddle, Matlab, SAS, and NNL by Sony. It defines converters for Keras, Tensorflow, scikit-learn, Apple Core ML, Spark ML, LightGBM, libsvm, XGBoost [17, 18].

There are many projects for machine learning on mobile, too. A popular usage of neural networks on Android is image recognition and manipulation, but these models can also be used to detect malware [19–21].

Using machine learning models through web services is quite popular nowadays, because there are many devices that cannot train a model by themselves, so they must access pretrained ones [22].

Distributed machine learning is a very popular topic, and there are many unique ways to achieve it. There are implementations with parameter servers, where the distributed computations update the model parameters on a defined server [23], and there are special systems like MLBase [24], what defines an own architecture for distribution, or MLI [25], what defines an API, or more widely used solutions, like Apache Spark's MLlib [26]. Another important branch of parallelization is model parallelism, which becomes particularly necessary when the number of model parameters exceeds the memory and computational capacity of a single device. A recent survey, published in 2024, provides a detailed analysis of different types of model parallelism, their implementation challenges, and modern applications, especially in the case of large-scale transformer architectures [27]. According to this study, model parallelism is essential for training large language and vision models, and it complements the possibilities offered by data parallelism.

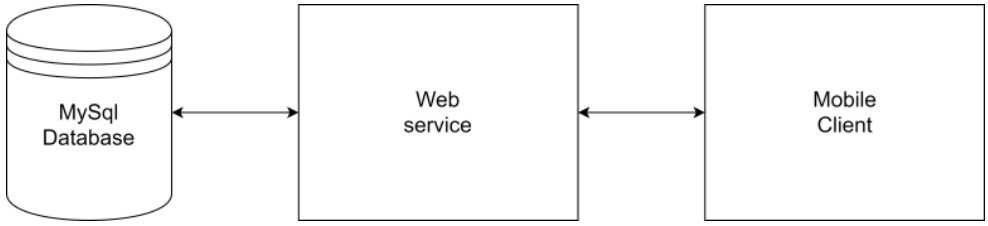
In addition to architectural frameworks, recent work has also focused on evaluating the efficiency and scalability of distributed and parallel approaches. A comparative study published in 2024 highlights the trade-offs between distributed and parallel machine learning, emphasizing their relative effectiveness in large-scale data processing tasks and providing practical benchmarks for real-world applications [28]. This comparison underlines the importance of selecting the appropriate strategy depending on data size, system resources, and deployment environment.

Beyond the architectural approaches, several studies have emphasized the practical limitations of model parallelism. Chatterjee provides an in-depth analysis of the implementation challenges, with particular attention to the significant communication overhead that often arises when models are split across multiple devices [29]. This highlights one of the main trade-offs of model parallelism: while it enables the training of very large models that cannot fit on a single device, the synchronization and data exchange between devices may considerably reduce overall efficiency.

2.2. Fundamentals of the System

The created software, which distributes machine learning on mobile, is shown in Figure 2.1. The center of this application is a Spring framework [30] based web service, which manages datasets and trained models too. It communicates with a database, which will contain the datasets and the models. The mobile devices can

communicate with this service, so they can get data slice, trained models, and ensemble models from it, and they can send their trained neural networks back.



2.1 The relationship of the system's components

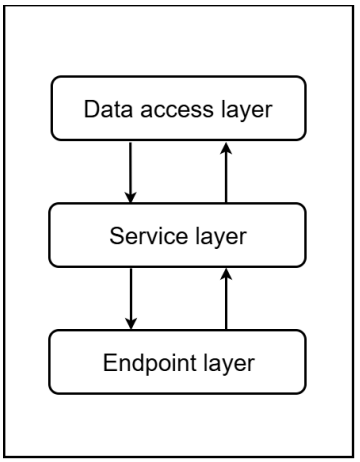
2.2.1. Web service

The web service is the most complicated part of this system because it includes methods for managing datasets, processing data, converting data, persisting trained models, retrieving trained models, creating ensemble models, and handling Hypertext Transfer Protocol (HTTP) requests. It is important that this part of the system was written in Java using the Spring framework. Of course, this software can be implemented in any programming language that can handle HTTP requests and responses and has a common data exchange format with the Android system. In our software, the data on both server and client is represented by a Java byte array.

The data access layer of the web service uses Spring Data to run SQL commands in the database. In this layer, there are two sublayers, named entities and repositories. Entities are the Java side representation of the database objects, so the Spring framework will know what kind of data it can access in the database. The repositories define interfaces containing operations like create, read, update, and delete, so the framework will use these to generate a complete query with the given entity types.

The service layer is in the middle of the web service software, so it communicates with the data access layer and with the endpoints' layer. It contains the business logic of the software, which includes many functions, like converting database entities to other data types, slicing bigger datasets into smaller ones, creating ensemble models, and forwarding database-related requests to the data access layer. The conversion of database entities is necessary because the structure of the objects in the database differs from the object structure needed by the mobile clients. The slicing of the dataset can be managed in many ways; for example, the user can request a large slice of the dataset containing random records. In this layer, there is a domain sublayer that defines the classes from the Android application, so the database entity can be converted to objects from the domain.

The top layer of the web service contains the endpoints and some of the functionalities. Classes defined here have similar tasks like handling HTTP requests, generating HTTP responses, and validating the given JSON web token. Besides that, they have unique tasks, too. The architecture of the web service is shown in Figure 2.2.



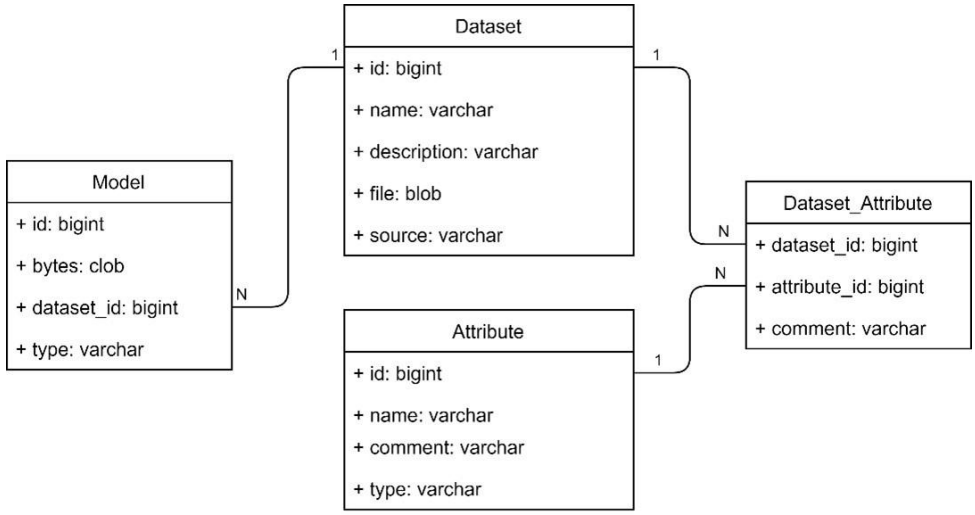
2.2 The layer structure of the web service

For example, with the ensemble endpoint, the user creates an ensemble model with the given type and the given models, or it can query persisted ensemble models from the database, so it can use these models on the client side. This layer has one of the core functionalities, which is the conversion of the models to a byte array, so it will be easier to send through the network, and it can be converted back to a model object on the client side. The list of defined endpoints is shown in Table 2.1.

2.1. Table - Endpoints of the web service

Prefix	Endpoint	Method	Description
/data	/slice	GET	Retrieve an n-size dataset slice
/data	/get-dataset	GET	Retrieve the whole dataset
/data	/get-dataset-attributes	GET	Retrieve the attributes of the given dataset
/data	/list-dataset	GET	Retrieve the list of available datasets
/data	/remove-dataset	DELETE	Remove the selected dataset
/model	/list-model	GET	Retrieve the model list for the given dataset
/model	/get-model	GET	Retrieve a single model by identifier
/model	/save-model	POST	Save the model specified in the body
/model	/remove-model	DELETE	Remove the selected model

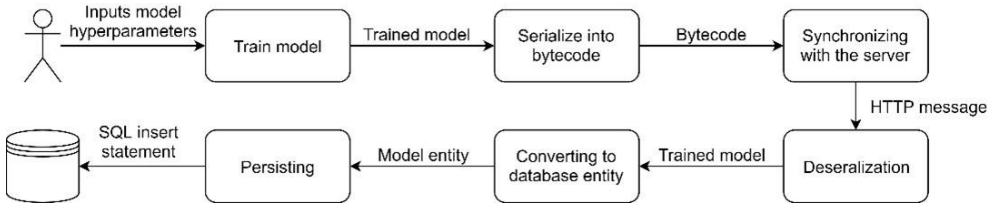
The functions of the web service software include dataset management, data processing, data conversion, dataset slicing, persisting trained models, retrieving trained models, and creating ensemble models. The dataset management function is about accessing the datasets, which are persisted in the database. Because of the variety of parameters the models can have, they must be persisted in a common form, which can be a byte array of them. The data model in Figure 2.3 shows that the database can work with multiple datasets by having a table for the attributes and the data types of them, and each record of this table references the related record of the dataset table. With this solution, the web service can query the database for dataset files, and the information for reading them can be accessed with a query to the attribute table.



2.3 Database structure

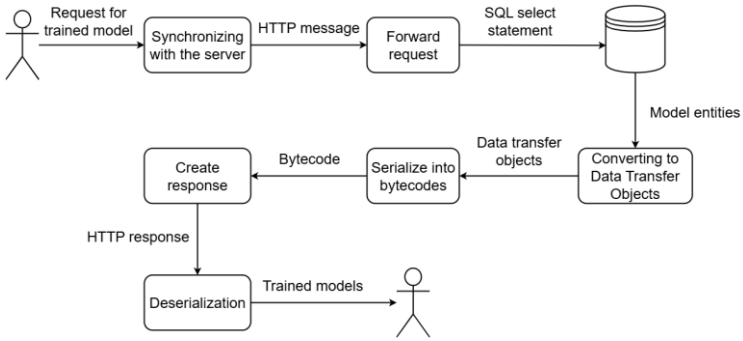
The data processing function is about preparing the queried dataset to be in the proper state for training. The transformation of the dataset can be customized, so the user can access the Deeplearning4J library's functions. We chose this library because in this chapter, we use Java-specific operations, and according to the documentation, it seemed easier to learn and use. The user can apply data reduction or can change the scale of the data; it can use sampling, dimensionality reduction, feature extraction, discretization, or binarization. These features use Deeplearning4J's corresponding functions, such as Principal Component Analysis (PCA). The data slicing function is about getting an n-size random sample from the preprocessed dataset, so it relies on the data processing package.

Model handling is the most important function of the web service, because the created software's main feature is the usage of trained models from the network. The first model handling operation was persisting, so the service can persist the models trained on mobile devices. Here, when a model arrives at the server, it will try to deserialize it from a byte array to a simple Java object. If this process is successful, then the software knows that the sent byte array is a valid model, so it can be converted into a database entity, and then it can be persisted too. The flow of the model data is shown in Figure 2.4.



2.4 Flow of the model data from mobile to database

The second model handling operation is retrieving, where the user can query for models from the database. When a specific HTTP request arrives at the server with the number of the model, it will query the database for it; then it will be serialized into a byte array and then sent back to the client, where the byte array can be deserialized into a simple Java object, which will be a trained model. Serialization is a quite simple process to create a compact form of a Java object. The initial state is when there is a trained model on the server, and this model has attributes for hyperparameters and for the state of the model. These values will form a JSON object, which can be handled as simple text. This Java text object can be transformed into a byte array, so it will be in a more compact and verifiable format, which can be used for model data exchange. The deserialization includes the same steps, just in the opposite way. The flow of the model data is shown in Figure 2.5.



2.5 Flow of the model data when retrieving a model

The last model handling operation is creating the ensemble models based on the earlier persisted models. The mobile client can send an HTTP request to get earlier persisted models for a dataset. From this list, the user can select models and then send another HTTP request with the models' IDs as parameters. With this data, the web service queries the models from the database, then creates a selected type of ensemble model with them. This model can be persisted in the database and will be sent back to the mobile client. The sending

works the same as sending regular models, so the ensemble model will be serialized into a byte array, and the mobile client can use it after deserialization.

2.2.2. Database

This part of the software consists of a simple relational MySQL database, which is created for managing trained models and datasets along with information about their data structure. The database model is shown in Figure 2.3. The most important table is the model, which contains the machine learning model's generated ID, the byte array of it, the dataset's ID, and the type of the model. In this representation, the user can select models by type or dataset. The related table is the dataset, which includes the name, source, file, and the description of it. The dataset consists of attributes, which are represented in the attribute table. It has an identifier, a name, a type, and a comment, so the user can search datasets by attributes. The relation between the attribute and the dataset table is represented in the dataset attribute joining table.

2.2.3. Mobile client

This part of the software consists of two layers, and it has the most important functionalities, like training and evaluating models. The main technologies used here are Retrofit for HTTP communication, Room database for a local database on Android, and Deeplearning4j for machine learning.

The first layer of the mobile client is created for the services. It is connected with the web service's endpoint layer and the client's presentation layer. The classes in the service layer define the core functionalities of the application, so the user can send HTTP requests and process responses from here and it can process the given dataset and train models on them. When the user of the application wants to use a trained model, it will call a function from the service layer to send a request to the web service, which will send back the byte array of the model. This byte array will be deserialized and then can be used as a trained model.

The presentation layer communicates with the user itself and the service layer, so it will react to the user's interactions and forward them to the service layer. Here, there are only files that define the user interface and the logic behind it.

The mobile client's main task is to train models and then send them back to the web service. For this task, each layer in the web service and the client will be used, so the data goes through the whole system. The first operation needed for training is retrieving the dataset, for which the client will use the web service's slice endpoint, which is created for sending dataset slices. When the client has the subdataset, the user can start training the chosen model from Deeplearning4j.

The model can be customized with a given seed number, optimization algorithm, learning rate, regularization, and, of course, custom layers can be added too. After the model training process, the trained model will be converted into a byte array and will be sent to the server, where it will be persisted.

Of course, there are a lot of limits for this mobile client because by default, Deeplearning4J was not created for mobiles, so it does not support many Android features. A fine example for this is that Deeplearning4J's model training and other operations are not thread safe, so it cannot utilize all of the mobile's resources, mostly the processor. The other problem with the mobile platform is the lack of memory, because machine learning often needs much memory, especially with bigger datasets. That is why the slice operation was introduced: mobile devices can train on smaller datasets, and the result can be synchronized with the server.

The trained models are not sent to the server automatically; it is a user choice. There is an evaluation process on the mobile client, so the user can check the created model. The scores of accuracy, precision, recall, and F1 can be written to the screen. With the evaluation, the user can see the confusion matrix, false positive/negative rate, true positive/negative, class counts, F-beta, G-measure, and Matthews correlation coefficient from Deeplearning4J's eval package.

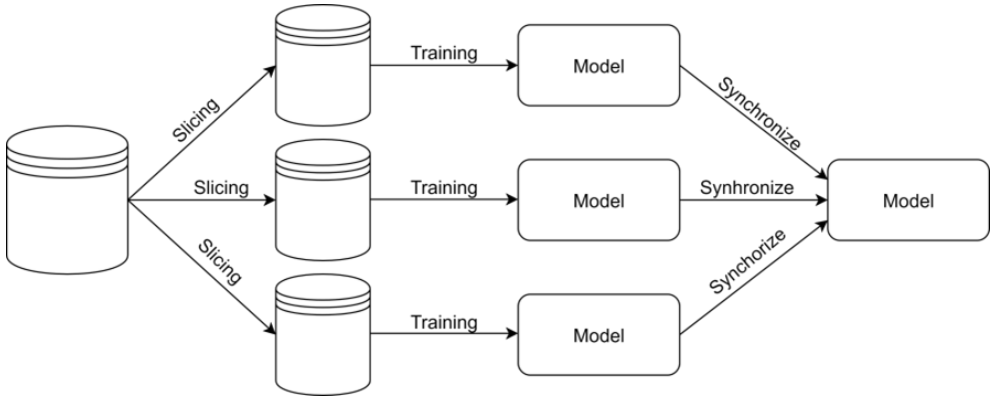
The last function of the mobile client is to reuse the trained models from the database to create ensemble models. This can be achieved from the user interface, where the user can retrieve the list of models for the selected dataset. For communication with the web service, the mobile application's service layer uses the list-model endpoint. Here, the user can select some models and then an ensemble method. When the ensemble model is created, it can be evaluated with the evaluation service. If the user finds the model good enough, it can be sent to the web service to persist it in the database. For saving, the mobile will use the web service's save-model endpoint. The persisted model can be reused later by other mobile devices.

2.3. Learning on Mobile with Data Parallelism

Most machine learning systems use some kind of parallelism when they have to deal with bigger datasets. They can be separated into model and data parallelism. In this chapter, model parallelism was not used, so this software is about utilizing mobile processors with a fully data-parallel machine learning system.

2.3.1. Data parallelism

Data parallelism is a popular way to introduce parallelism to software because it is fairly easy to implement, and it is independent from the chosen machine learning model. Data parallelism is not only for machine learning, although it is obvious to use it for handling big data or reducing computation needs [31, 32]. Basically, this method is about having one bigger dataset, which will be used for machine learning, and from it, multiple subdatasets can be created. It is important to note that if the size of the subdataset is too small, it possibly contains too little information for a machine learning system, but the current system does not warn the user about the dataset size. These smaller datasets are easier to process, so they can be processed on separate processors or devices. The result of the training on the subdataset will be a smaller model; it can be named as a submodel, which contains information about the subdataset. These submodels can be synchronized to a server for aggregation. The output of the synchronization process will be a bigger model combined from the submodels, so it will contain information about each of the smaller subdatasets. The basic data parallelism can be seen in Figure 2.6.

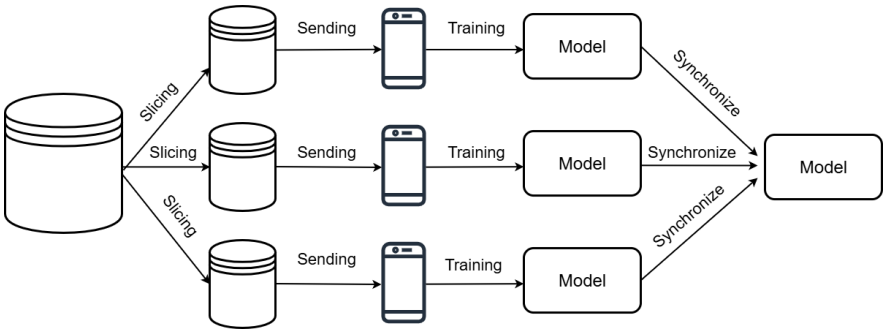


2.6 The simplified data parallelism

2.3.2. Data parallelism with mobile

Using data parallelism with mobile is only an extension of the basic data parallelism, so the main steps are the same. The problem here is handling bigger datasets too, but the computing capacity here is in the mobile devices, not simple processors. The first step is the slicing of the data, which will result in multiple smaller datasets. While creating smaller datasets, it is more important to care about the size of the resulting datasets, because besides the lower size limit, the mobile devices have an upper size limit too. If the subdataset is too large, the mobile device cannot process it because of the lack of memory in these devices.

With the introduction of mobile devices, the subdatasets will be sent to the mobiles through the network. The training process will start here on these devices and when the training is over, the model can be sent back to the server with the synchronization process, where they can be aggregated. The output here will be a bigger model combined from the submodels, just as in simple data parallelism. So basically, this method involves the new synchronization process for the mobile, but with it, the summed computing capacity will be bigger, because mobile processors can run training processes on each core. The mobile extension of data parallelism is shown in Figure 2.7.



2.7 Extension of data parallelism to mobile

2.4. Synchronization

The synchronization had many problems, which had to be solved. The biggest one was that to use the trained model on the server, the client and the server should use the same machine learning library. This compatibility problem was solved with Deeplearning4J, because it is written in Java, so Android devices and web services can use it too. The other problem was that sending a complex model through the network can be problematic. The solution for this was the serialization of models to a byte array, so they act like compressed messages which can be deserialized on the other side. The serialization process starts with the trained model, which is a Java object that can be converted into a JSON object. This object includes the hyperparameters and the state of the model in a special text format. This text object will be converted into a byte array, because it is more compact and easier to verify. When software uses a network to send and retrieve data, it has to prepare for data loss. The message can arrive malformed, partially missing, or it's possible that it does not even arrive. The byte array serialization is a solution for this problem, too, because if the byte array is malformed or partially missing, it cannot be deserialized on the other side, so the software will know about the error. The last discovered problem of

the serialization is that its performance depends on the network speed, so sending models and data will be slower on slower networks.

2.5. Measurements

An implementation was created for the mentioned software architecture, so measurements can be made with it. The web service used Spring Boot, Hibernate, and Deeplearning4J to implement the endpoints, and the mobile client used Room database, Retrofit, and Deeplearning4J. For comparing the training time on a simple computer and on this software, the dataset was 'Record Linkage Comparison Patterns Data Set' from the Epidemiological Cancer Registry of North Rhine-Westphalia and the UCI Machine learning repository [33, 34]. This is a simple dataset for classification with 12 attributes and 5749132 records. The reason for choosing this is that the number of records is large, and it was certain that it could not be processed on mobile devices without slicing. The dataset is about comparing two records based on the agreement of first name, family name, sex, and the date of birth's day, month, and year components separately. The same Deeplearning4J code ran on the desktop and on mobile, too, so they trained the network with the same parameters. The evaluation of the desktop and mobile models shows that both of them had nearly the same results as seen in Table 2.2. This accuracy was computed for a 10000 large subdataset.

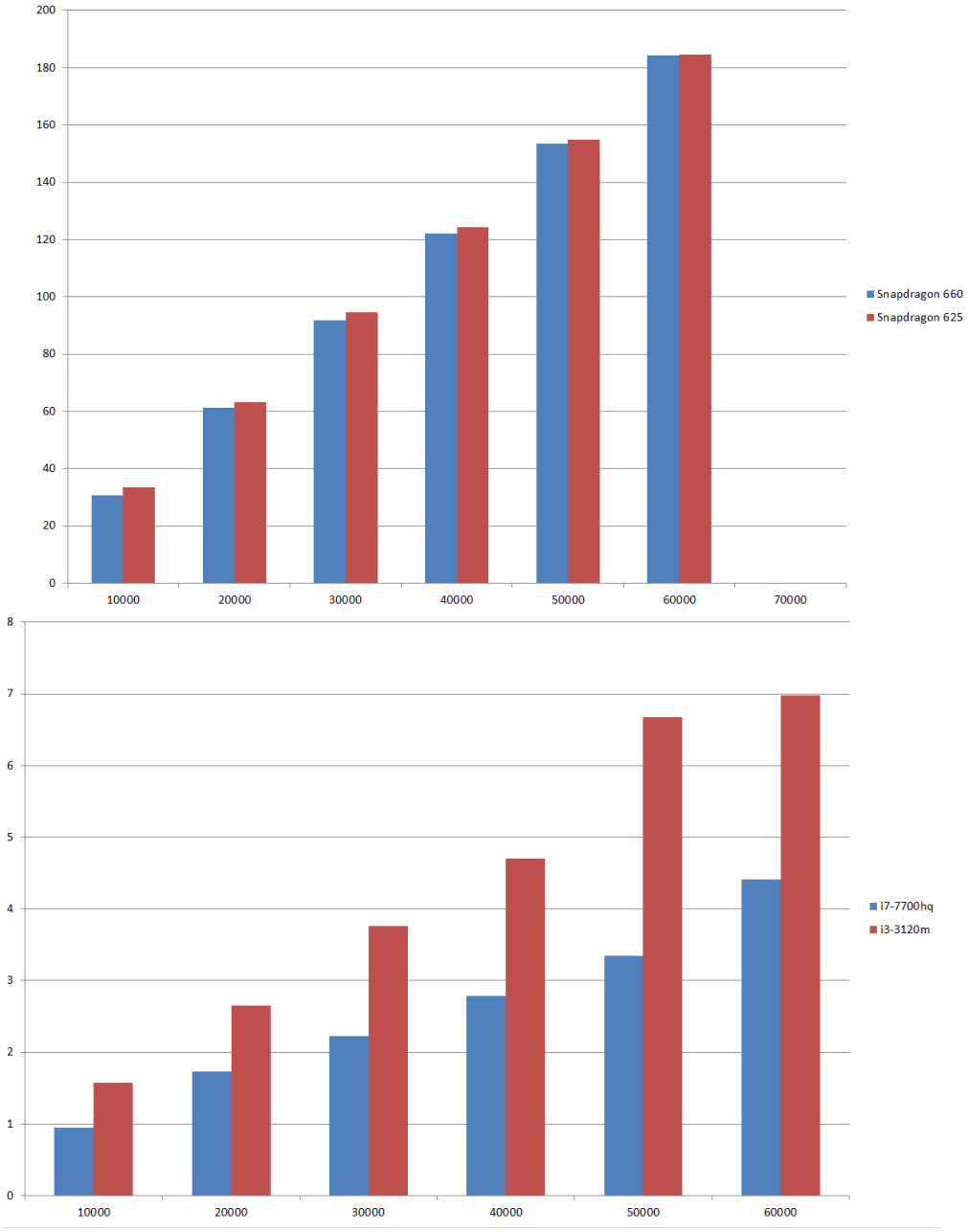
2.2. Table - Accuracy and best score on mobile and computer

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

The accuracy of the submodels is lower than that of the bigger model, which was trained on the entire dataset; however, ensemble models constructed from the smaller ones have as good accuracy as the bigger model. In numbers, the accuracies were about 0.79 for submodels, 0.99 for the ensemble, and the model trained on the full dataset.

The creators of Deeplearning4J noted that this library can show weak performance on Android, and that was confirmed by my measurements too. The concrete runtime of 1 epoch based on the size of the dataset can be seen in Figure 2.8. The X-axis shows the number of records in the dataset slice, and the Y-axis shows the runtime of 1 epoch in seconds. The figure shows that these mobile processors can handle datasets containing fewer than 70000 records. In Figure 2.8(a), it can be seen that the training speed is quite slow on the mobile platform with the given Deeplearning4J version, so it can process only smaller

datasets effectively. Figure 2.8(b) shows that the training speed is quite faster on a desktop environment.



2.8 Runtime of 1 epoch based on the size of the dataset on a) mobile and b) desktop

2.6. Possible Improvements and Conclusion

The measurements showed that the training speed with Deeplearning4J on mobile as of 1.0.0-beta5 is quite slow and is limited by the non-thread-safe machine learning library; however, the created models' accuracy, precision, and recall values were similar to the simple models. The Android machine learning support mainly focuses on running pretrained models rather than training on mobile processors, so its support is very limited. Overall, it can be seen that the data parallelism on mobile can work well, so these devices can be part of a cluster environment for training models. The architecture is easy to implement and it was created to be extendible.

It is important to note that because of the web service that handles all of the models and the data, additional methods should be added to ensure the security of the system. It could be extended with some kind of role based access control.

The implementation could be improved in many ways, for example the slow training process can be solved with another machine learning library, which is available on mobile too, so it should be changed, maybe for Tensorflow.

The architecture itself can improve to support the multiplatform distribution, so training could be done either on mobile or a simple processor, the results can be aggregated. This may increase the computing capacity, but it can generate a massive load on the server. This load could be handled with redesigning the architecture to support microservices with multiple servers.

2.7. Relevant thesis

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.

Chapter 3

Resource-Efficient Machine Learning on Android: A Case Study of Weka, Tribuo, and SMILE

This chapter is based on my publication, entitled Machine learning on Android with Oracle Tribuo, Smile, and Weka [O2].

In September 2020, Oracle announced Tribuo, its open-source Java machine learning library under the Apache 2.0 license. It features many commonly used algorithms like random forest, SVM, lasso, K-means, so it can solve classification, regression, clustering, and anomaly detection problems. SMILE, the Statistical Machine Intelligence and Learning Engine, is another machine learning library for Java. Its main advantage is performance compared to other libraries and algorithm support. Weka is a general-purpose open-source machine learning software with a Java API, which is easy to use and has its own graphical interface. The common thing in these libraries is that they can be used with Java or Kotlin and there are many algorithms that all of them support. Because of this, their performance can be compared in the same environment, which will be an Android device with a mobile processor in it. Although these are not native Android libraries, they can work on these systems, and their performance can be compared. Kotlin is a programming language for the JVM, which has become the preferred language for Android programming. Codes from Java can be transformed into Kotlin code, so it is easy to use Java libraries in this environment. Benchmarking mobile devices' model training performance is a repeating task, because we can measure how much these devices have evolved over the years. In this chapter, we present the Android machine learning ecosystem, the libraries, the challenge of porting machine learning libraries, and the results.

3.1. Related work

With the evolution of mobile devices and applications, it was inevitable to use machine learning techniques for a more personal user experience. Most applications use pre-trained models to recognize voice, to take better pictures, or to swap faces. There are many disadvantages of training models on mobile, for example, the energy consumption [35]. Besides that, there are many use cases of models trained on mobiles, like comparison of machine learning capability of processors [36], detecting potholes [37], or malware [24]. This chapter can be placed on the topic of machine learning and Android benchmarking. There are

many articles about comparing devices or machine learning libraries by training time, memory, and CPU efficiency. For a complete benchmarking tool, there is the PMLB, the Penn machine learning benchmark [38]. There are other papers about benchmarking, like the Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark [39], the MLPerf Training Benchmark [40], or the Benchmark of Machine Learning Methods for Classification of a Sentinel-2 Image [41]. A recent survey provides a comprehensive overview of existing on-device training systems, highlighting their architectural characteristics, common design patterns, and the key challenges posed by limited memory and energy resources [42]. This work serves as a valuable reference point for understanding current approaches to efficient training directly on mobile devices. The NNTrainer v3 framework demonstrates that on-device training and personalization can be achieved with significantly reduced memory overhead, while maintaining competitive accuracy through system-level optimizations tailored for mobile platforms [43].

3.1.1. Java machine learning

Java is not a popular language for machine learning, but it is popular for application development, and because of the need for intelligent applications, it supports many machine learning features with libraries. Each of these supports different algorithms and datasets, and each of them has advantages for specific systems. Weka [44] is a complete machine learning software with a graphical interface, a command-line interface, and it can be used as a Java library too. It supports tasks like preprocessing, classification, regression, clustering, association rules, and visualization.

SMILE [45], the Statistical Machine Intelligence and Learning Engine, is a powerful engine that covers every aspect of machine learning. It supports JVM languages, so SMILE code can be transformed into Kotlin code. It has many algorithms for classification, regression, clustering, association rule mining, and many built-in solutions for pre-processing, validation, feature engineering, and time series.

Oracle Tribuo [46] is a newly open-sourced machine learning library written in Java, and it has unique features like provenance, type safety, and interoperability. Models, datasets, and evaluations have provenance, which means they know the transformations and parameters used to create them. The interoperability means that Tribuo has interfaces to libraries like XGBoost [47] and Tensorflow [48] and the ONNX [18] exchange format. Deeplearning4J [49] is a SMILE-based library focusing on deep learning, which is not in the scope of this chapter. H2O [50] is an in-memory platform that has many software and libraries for machine learning. It supports most kinds of data mining tasks and it

can be integrated with Java applications through REST API or embedding. Mallet [51] is another option for applications that use natural language processing, document classification, or clustering.

3.2. Machine learning on Android

When we want to train and use machine learning models on Android, we can encounter many challenges. The main reason not to train models, especially with large datasets, on mobiles is that these operations need a lot of energy, so running applications would result in battery drain. While training is possible, it is limited by these devices' memory capacity, because most mid-range smartphones usually have 4-6 gigabytes of memory, and this is not enough for processing larger datasets, not to mention that Android has limitations for memory usage. Another challenge for mobile machine learning developers is the architectural difference between the processors of computers and mobiles. Using Java libraries on these devices can be risky because Android does not have full Java support, so it is possible that some functions do not work or work, but with different results.

Currently, applications that want to use machine learning can choose from using TensorFlow Lite, or a library with Android Neural Network API support, or web services. With TensorFlow Lite, developers can mostly use pre-trained models created with TensorFlow, or they can train specific models for image and text classification. A popular choice is to use machine learning web services, where the application sends data to the service and gets back the result.

3.2.1. Porting Weka

Weka contains many Android-incompatible code, mostly its graphical interface and logging, but there are many more functions that use specific code parts, which cannot be compiled on mobiles. Our strategy here was to remove everything that is incompatible and see how the application can work with the newly compiled Weka. The porting was successful; however, sometimes minor errors occurred during the tests. The tested library was based on rjmarsan's Weka-for-Android project [52].

3.2.2. Porting SMILE

SMILE also contained incompatible code, such as Java code that was not supported by Android or functions with the java.sql type. Because the number of these code parts was few, they were replaced by Android-compatible ones. The result was good enough to run on mobiles, but it did not support all features, and some of these errors occurred during runtime.

3.2.3. Porting Tribuo

Oracle's Tribuo library consists of many Maven artifacts, and some of them were not needed for this application. There are modules which cannot be compiled on Android, but the most important ones, namely tribuo-data, tribuo-classification-trees, tribuo-clustering-kmeans, and tribuo-classification-sgd, ran without modifications. Overall, we can say that not all of Tribuo's functions work on Android, especially the ones using third-party libraries, but we can do simple machine learning tasks with it on mobiles.

3.3. Application

The aim of the Android application is to measure properties of machine learning libraries, such as memory usage or runtime. The libraries involved are the newly open-sourced Oracle Tribuo, SMILE, and Weka. The application runs the same test with each library, which means it trains models like SVM, Random Forest, and K-means on multiple datasets with different sizes. The test uses the same algorithms and parameters for all libraries. The application logs the results and runtime properties.

The graphical interface is quite simple. When we tap on a library name, the software will train a selected type of model on a selected dataset. The "ALL" button is for running the training operation of all libraries in parallel. We can choose from the Iris dataset [53] with 150 records, a subset of the Record Linkage Comparison Patterns dataset [33][34] with 60,000 records, and the full Record Linkage Comparison Patterns dataset with 5749132 records. The algorithms we can choose from are the random forest with 500 trees, split rule GINI, maximum depth = 20, maximum nodes = data size / 5 and node size = 5, the SVM with an RBF kernel, gamma = 0.1, lambda = 0.5, epochs = 30, and the K-Means algorithm with 2 or 3 centroids based on the dataset, iterations = 10 and distance is Euclidean. The output result contains the runtime, the maximum and the average CPU usage, the memory usage, and the energy consumption.

3.1 First screen of the measuring application. The user can decide to run tests for a specific library or all available libraries.

3.4. Results

Weka, SMILE, and Tribuo can train machine learning models on Android, and all of them can be used for simpler applications. As the software trained the same models with the same parameters and datasets these models' performance was the same on the test datasets. The only difference between these libraries was in the runtime, memory usage, average CPU usage and energy consumption.

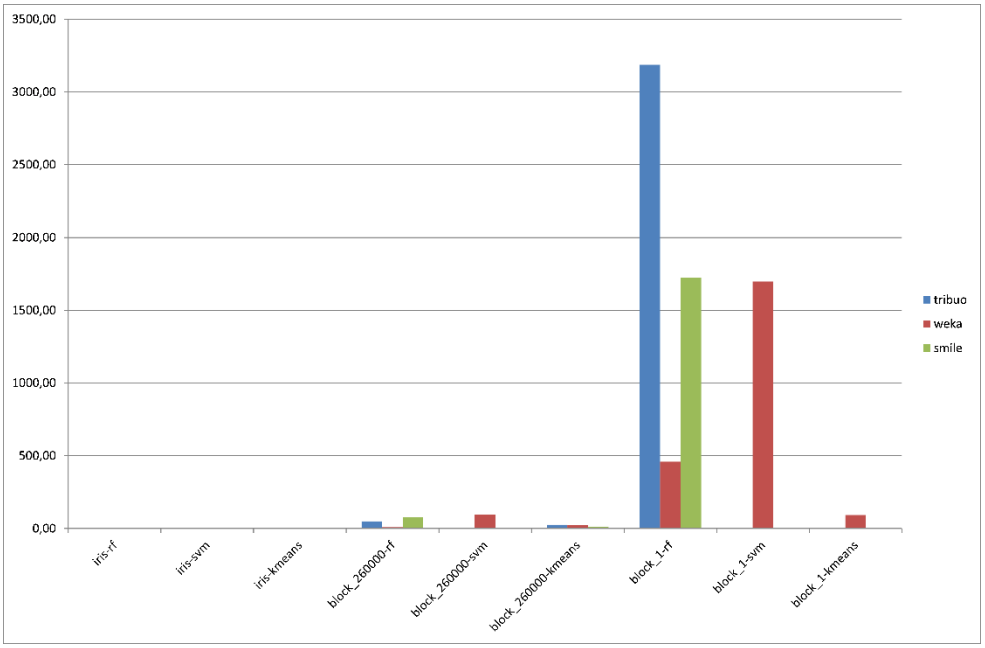
3.4.1. Runtime

In Table 3.1 and Figure 3.2, we can see how fast the libraries finished the training of the models, where the prefix is the dataset (i is the Iris dataset and p is the Patterns dataset), and the second part is the name of the algorithm (where r is

the random forest, s is the support-vector machine and k is the K-means). For Patters, the prefix 6 means 60000 samples from the full dataset.

3.1. Table - Runtime of algorithms on specified datasets.

	i-r	i-s	i-k	p-6-r	p-6-s	p-6-k	p-r	p-s	p-k
Tribuo	0.9	1.1	0.5	49.7	0	25.3	3187.5	0	0
Weka	0.7	0.7	0.7	9.3	95.8	22.7	459.3	1696	92.8
SMILE	3	1.4	0.1	75.6	0	13.2	1725	0	0



3.2 Comparison diagram of runtime results.

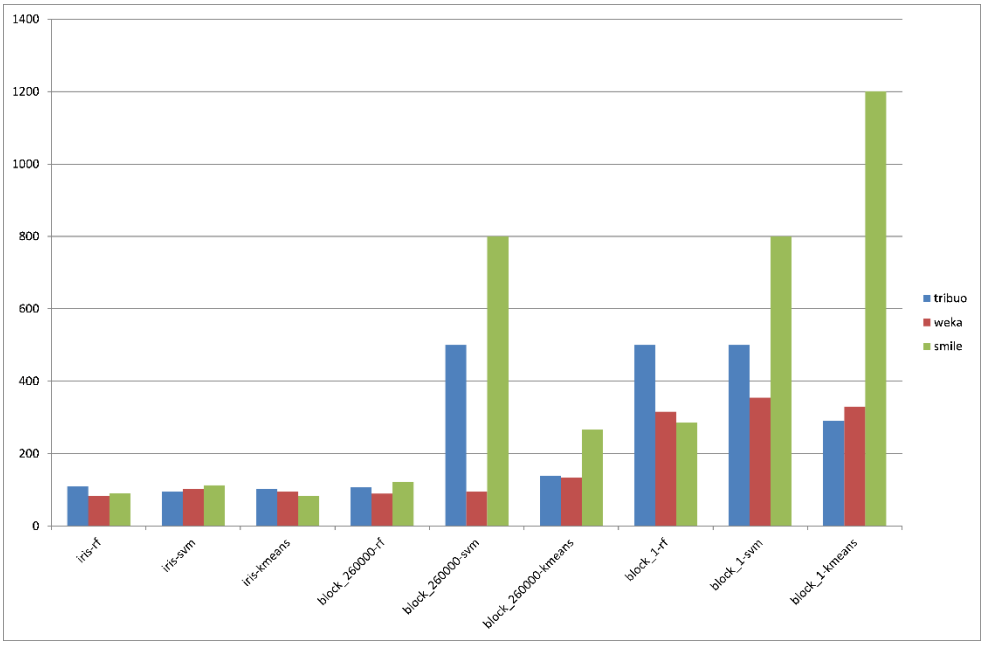
As we can see, most of the time, Weka library was the fastest, except for K-means, where SMILE was faster. For smaller datasets, both Tribuo's and SMILE's results are good, but for larger ones, they were much slower than Weka. Where there are zeros in the table, the software did not complete the training of the model, because the Android system killed the application due to its high resource needs.

3.4.2. Memory consumption

In Table 3.2 and Figure 3.3, we can see how much memory the algorithms needed in megabytes. The notation is the same as in Table 3.1 and Figure 3.2.

3.2. Table - Memory needs of algorithms on specified datasets.

	i-r	i-s	i-k	p-6-r	p-6-s	p-6-k	p-r	p-s	p-k
Tribuo	110	95	102	106.3	500	139	500	500	290
Weka	82.5	103.4	96	89.3	95	133.8	316	354.6	330
SMILE	90.5	113	84	121.7	800	265.8	285.4	800	1200



3.3 Comparison diagram of memory results.

It is clear that Weka used the least amount of memory, but in some cases, Tribuo was very close to it, like in iris-svm, iris-kmeans, and patterns-kmeans. There are special cases where the training did not finish, like Tribuo's pattern-60000-svm, pattern-rsvm, and pattern-kmeans, or SMILE's pattern-60000-svm, patternsvm, and pattern-kmeans, where the memory needed was extremely high compared to other cases. The highest value was the SMILE's K-means training for the Pattern dataset, where the application used 1.2 GB of memory. For smaller datasets, we can say that Tribuo and SMILE needed somewhat more memory, but this is not a huge difference.

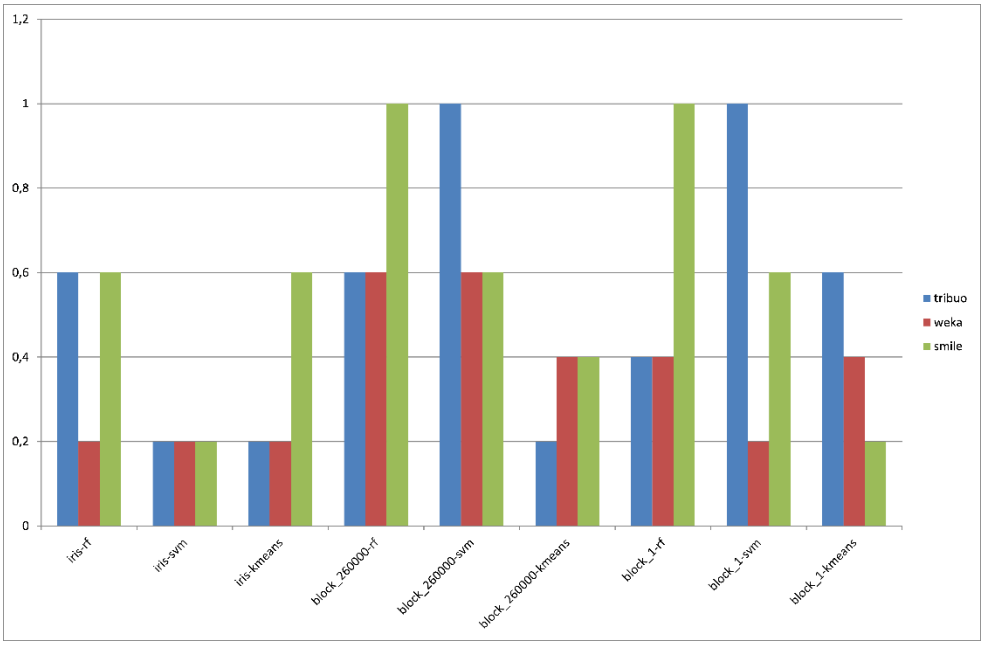
3.4.3. Battery consumption

In Table 3.3 and Figure 3.4, we can see the battery consumption of the application when it trains the selected models. The table's and figure's notations

are the same as earlier. The values range from 0 to 1, where 0 is no energy needed and 1 is the highest energy need.

3.3. Table - Energy consumption of algorithms on 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



3.4 Comparison diagram of energy consumption results.

Battery consumption is an important part of these measurements, because this means that a machine learning application could be maintained, or it drains the battery so much that the application is unusable. In Figure 3.4, we can see that Weka used the least amount of battery, the next one was Tribuo, and the hungriest library was SMILE. For smaller tasks, Tribuo's and Weka's energy needs were nearly the same.

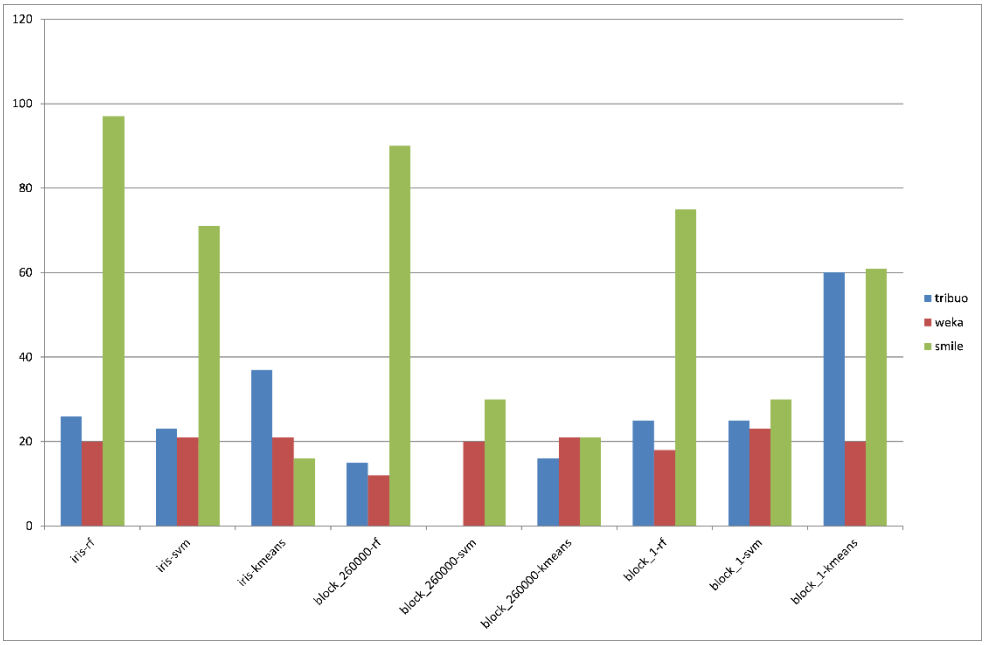
3.4.4. Average CPU usage

In Table 3.4 and Figure 3.5, we can see the average CPU usage while training the models. The table's and figure's notations are the same as earlier.

Average CPU usage shows how these libraries use this resource. When this value is below 50, it means that other applications can run in parallel while the training is running.

3.4. Table - CPU usage of algorithms on 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



3.5 Comparison diagram of CPU usage results.

Table 3.4 shows that SMILE needs the most CPU resources for every algorithm and dataset. The second most processor-hungry library was Tribuo, but it produced results similar to those of Weka. This means that with better processors, SMILE would perform better in runtime. For weaker processors, it is strongly advised to use Weka because of its small resource needs.

3.5. Conclusion

This chapter showed that using machine learning libraries meant for Java can be used in Android application development if we have a specific task.

However, if an application uses models for image or text classification, the recommendation is to use Tensorflow or web services. Specific tasks can be classifying or recommendation-based on the users' data, where it is necessary to train a local model.

Porting Weka, Tribuo, and SMILE to Android devices is a somewhat challenging task, because each of them has some Java version or platform-specific code, which will not work on mobiles. To compare these libraries' performance, we must select algorithms that each of them supports and has an Android-compatible implementation. For the comparison, we chose the random forest, SVM, and K-means models and datasets with different sizes. The results show that Weka is the suggested library for bigger datasets and complex models, as it is the least resource-hungry. It supports a wide range of algorithms, so every kind of machine learning task can be done with it. For smaller datasets or less complex models, Tribuo and Smile can be an option because they get updates frequently and can react faster to market needs.

An improvement can be extending these tests with other datasets, algorithms, or multiple devices. The results could be compared to results from computers instead of Android devices. As new machine learning libraries are released, they could be ported to mobiles, and the results could be extended.

3.6. Relevant thesis

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.

Chapter 4

Designing Scalable Ensemble Learning Systems Using Web Services in a Microservice Framework

This chapter is based on my publication, entitled Building ensemble models with web services on microservice architecture [O3].

When we train machine learning models, it is often done in our favored programming language; however, it might be complicated to integrate them into an application. The easiest scenario is when the model is stored in the application, which means they are possibly written in the same language and the machine learning library is supported. In this case, we can simply use it, for example, to make predictions or recognize images. When it is not supported or usable by our preferred language, the simplest solution is to have a different application just for the machine learning tasks, and it will communicate with the original one. This solution requires extra resources for communication, as the model or results have to be sent over the network. This first scenario can cover applications built on monolith architecture; the second one can cover the layered architecture. Of course, these are just the obvious examples of integrating machine learning in our web applications. Cloud services can be an option, but their results are also queried through web services, so it is similar to the second case. When we want to integrate machine learning into applications, we have to decide what kind of algorithms we would like to try and how to access them. When we have the answer for accessing the models from applications, our project is already restricted to a given set of libraries. For example, if we choose to make a machine learning module that can be accessed by the application directly, our project should use libraries of the same language. The other question is what kind of models would be trained? Is deep learning needed, or is K-nearest neighbour enough, or is the data suitable for unsupervised or supervised learning? After decisions like these, the set of available libraries is narrowed. In most cases, this is enough to choose, but if we want to use more than just one of them, because one of them supports algorithms that the other does not, we might encounter some trouble. In this case, the applications have to communicate even more.

Microservice architecture structures the application as a collection of services that are independently deployable and loosely coupled. Each microservice consists of a data store and application logic, which can only be accessed through its public API. As these software components are independent,

each of them can use proper tools and languages to achieve their goals, so it is possible that one component is written in Python, while others can use Java if it is suitable for their tasks. As a microservice is a small part of the whole application, it can be developed or maintained by a smaller developer team, and as it is independent, they can choose their own technology stack. Of course, this architectural style is not applicable to any kind of program, because the huge number of HTTP requests can be costly. The main disadvantage can be the communication between the microservices, because if the application is big enough and contains several services, the number of requests and responses can increase. In a standard web application environment, that means that the program packages the data in JSON format, sends an HTTP request, the other service receives the message, extracts the JSON to its object type, and then writes and sends a response, which is processed by the sender. This messaging can occur even ten thousand times per second. Besides this, we can mention latency, bandwidth, or topology that can affect the performance of microservice-based software. The importance of this architectural style can be seen from Google Trends [54]. Many software projects use it because development time can be shortened as the tasks are much smaller and easier to implement. Besides that, the testing of these services is easier because they are independent, and the codebase is much smaller than that of traditional architectures. The implementation can be written in any programming language, as it can be started in small steps. For example, we can create three web services that communicate with each other, and each of them has its own functionality. I chose a hybrid solution with the Spring Boot framework's reference implementation. The needed Maven dependencies are spring-boot-starter-web, spring-cloud-starter, and spring-cloud-starter-eureka-server. The Eureka server is used to register services so they can discover each other from there. The other dependencies are for creating web services, which can accept or send HTTP requests.

This chapter is restricted to web applications, as these are the most popular kinds of applications according to the JetBrains Developer Ecosystem Survey 2024 [55]. There are many ways to structure applications, as mentioned earlier, but we would like to achieve a machine learning system that connects different libraries from different environments, so the product could be language independent. The idea is that libraries can be wrapped in web services. To support the high number of them, a lot of services should be created, and to manage them as one application, we decided to use microservice architecture. It is the best decision as the elements are loosely coupled in the software, and data management can be decentralized, so services can send requests to other services through their public API, and each of them has the ability to manage its own database. In this way, multiple datasets and their models can be handled. For

example, we have a web service that wraps the Deeplearning4j library, so we can train models and query the results with HTTP requests. To manage data, train models, and extract the results, we only have to design the appropriate endpoints. When the web services are finished, the application can be assembled with the microservices.

The combination of different machine learning models is not rare, as many research state that it can improve performance. There are many ensemble techniques like bagging, boosting, voting, and stacking. The idea of them is that they mostly use only the results of the different models, and because of this, we can get these results from different platforms or services. One of the easiest is voting, which can be a majority or weighted vote. Diettrich's results show that in certain cases, ensemble models can outperform single classifiers [56].

The combination of machine learning and microservice architecture gives us the opportunity to distribute model training in both model and data parallel ways, and besides that, it has the advantage of connecting different platforms in one application. As disadvantages, we can mention the dependence on the network and the high number of communications. According to Google Trends [54], the keyword Microservice has been a popular search phrase since 2016; it was at the top in 2020, nowadays the number of related searches has decreased to 60 on a scale of 0-100. Although federated learning architecture also has a centralized and decentralized variant like this chapter, the process of training and the goal are somewhat different. Federated learning often results in a bigger model, because the local clients train models and update the global model's parameters [57]. In this solution, the client can choose between using single classifiers and ensemble models. The direct and gateway approach can align with any kind of client application, which can decide to put the computing load on the server or on its own side. We don't have parameter synchronization or parameter sharing; each model is independent. The commonality between federated learning and this solution is that both train local models, and both can end up in a single classifier.

4.1. Related work

There are many papers about connecting microservices and machine learning, but the purpose and the implementation can be different. Pahl and Loipfinger defined machine learning as a service, and their system encapsulated ML in a microservice with a REST interface [58]. DroidAutoML [59] uses the same architecture, but its goal is extended to mobile devices as it can configure and evaluate algorithms to detect Android malware. In this approach, the whole ML work is decomposed into small services, like model training, feature

database, scanner, and apk database, which then communicate with each other. Ribeiro et al. use a similar solution for machine learning deployment [60]. They proposed an architecture to support generic ML pipelines and implemented case studies to test it. This chapter uses ensembles to make predictions with different models, just like Attota et al. in their work [61]. They use several API gateways with model ensembling, and these gateways communicate with the central server's aggregation service. The cCube architecture [62] has a similar purpose: to easily develop and deploy ML applications, and for this, it uses an orchestrator service. The user communicates only with this, and it forwards the data to the learners and the scheduler. Our chapter presents a centralized solution that is based on this work. This technology can be applied in fields like healthcare, where KETOS [63] uses machine learning as a service, and the users can use R, Python, and DataShield in one application. It is a clinical decision support system with its own development and deployment process. There is a special use case of machine learning in software development, where we provision and classify microservices with these algorithms, like in [64], [65], [66]. For designing the architectures, we used the Guidelines for adopting frontend architectures and patterns in microservices-based systems [67].

Although it seems like microservices are perfectly suitable for most kinds of applications, they have several drawbacks and can cause many issues during development. It can increase the system's complexity because of the increasing number of independent services. This means that the project needs skilled developers. The architecture also has challenges in the areas of monitoring, versioning, and state management. [68]

Distributed machine learning also has its disadvantages, such as the increase in aggregate processing time and the need to work with multiple computing nodes. Because the training is more intense, it needs multiple computers, leading to increased energy consumption. Even with more computing resources, because of the distribution, it is not guaranteed that the training will be much faster. [69]

Recent surveys emphasize that machine learning is increasingly used not only for application-level functionality but also throughout the microservices life-cycle itself. A systematic mapping study from 2025 [70] shows how AI/ML techniques support autoscaling, monitoring, and placement of microservices. This underlines that the synergy between microservices and machine learning goes beyond model serving, extending to the optimization of the underlying service infrastructure.

Complementary to these architectural approaches, a recent comparative study [71] investigated the cost and performance trade-offs of different model-serving frameworks, including TensorFlow Serving, TorchServe, MLServer,

MLflow, and BentoML. Their results highlight that the choice of serving stack has a significant impact on latency, throughput, and operational cost, which is an important consideration when integrating ensemble models in a microservice-based system.

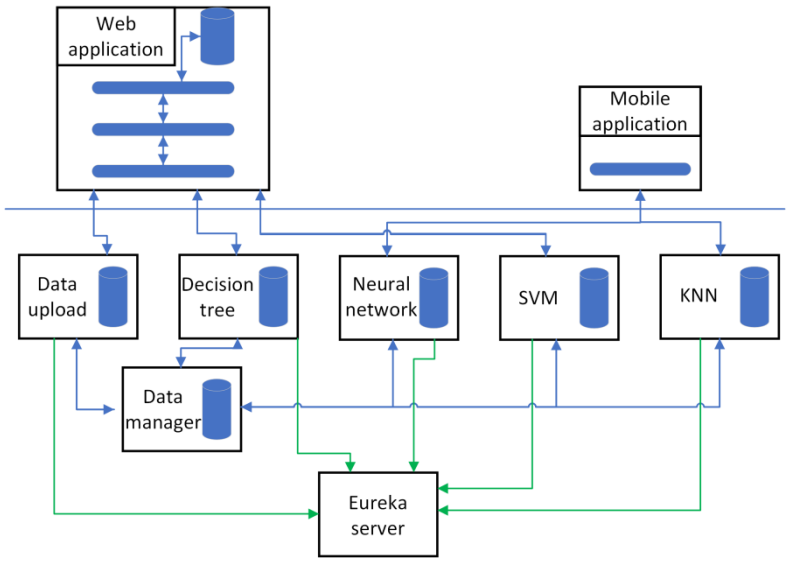
4.2. Architecture of the system

We had many ideas about how to integrate machine learning services in a microservice architecture-based application, but we narrowed it down to 2 variants: the gateway and the direct variant. The first one is a tree-like approach where the smaller services end up in a bigger summing service, and this works as if the application had a machine learning subsystem. This solution is similar to the MapReduce architecture, but it is specific to web applications. In a direct way, there is no such summarizing service, so the requester can process the results and, if they want, can combine them. In either of them, ML microservices should build models on the same datasets, so it is tempting to have one common database behind them, because if not, we must store the same data multiple times within one application. Rules of architecture forbid this solution because we would lose the independence of microservices, so all of them would have their own database. There are many patterns for building applications with microservice architecture, like the mentioned API gateway [67] and backends for frontends [67], and besides them, we use the gateway aggregation pattern from [72], [73]. Because of this pattern, we will often use the aggregator service and API gateway terms instead of each other. In our reference implementations, we built a backend system with Spring Boot, Spring Cloud with the Eureka Server for service discovery [74]. This server manages the services' IP address and port number, so this is where each client knows the addresses to which they can send requests. The Java microservices for training and managing Weka [44] and Deeplearning4j [49] models are written in the same environment. The Python microservices for Scikit [75], TensorFlow [48], and Microsoft Azure models used Flask [76] to be accessible by other web services.

4.2.1. Direct variant

In this variant, there is no API gateway, so we cannot use the backends for frontends pattern. As there is no gateway, the clients can directly access the models and can request predictions for their data. The combination of the trained models is not easy in this variant, because we do not have an independent node that creates ensemble models; instead, the client has to do it. The client can query the results of the model endpoints, and based on these, it can construct its own ensemble model. The advantage of this variant is that the clients can process

as little data as needed, because when they need a decision tree, they can query for one, when they need a voting or averaging-based ensemble method for two models, they query for the models' results and use the chosen technique. As there is no aggregator service, each of the microservices remains independent and does not rely on other machine learning endpoints' data, so the software remains loosely coupled, which is an advantage in microservice architecture. Integrating this solution into a backend system could be challenging, as there is no aggregator service, so the caller service must explicitly name a model service to use. We suggest this variant for applications that use machine learning for a few of their functions. In these systems, developers can use only the needed model from the service, so the network load will be distributed between them. As few of the ML services are used in the application, it should not cause maintenance difficulties. An implementation of this variant can be seen in Figure 4.1. The blue arrows indicate the path of the data; the green ones are for service registration. In this figure, we can see a web and a mobile application that uses different machine learning models' results using its web service.

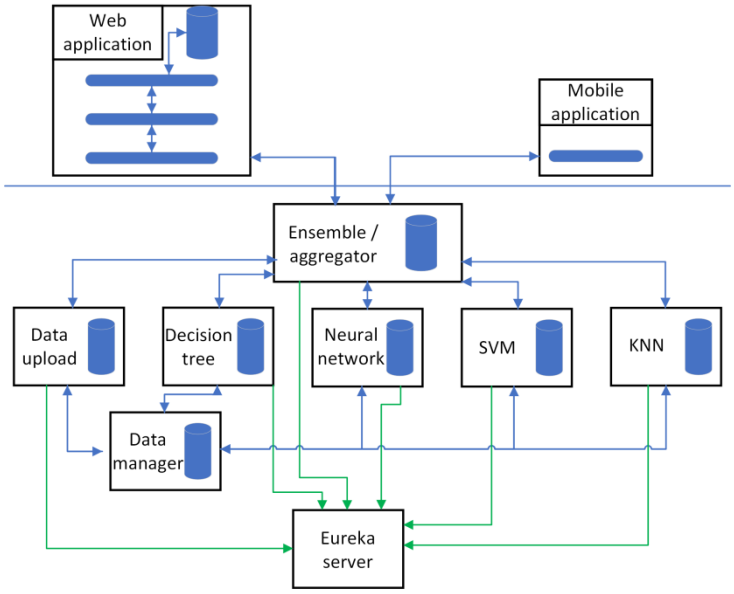


4.1 Direct architecture variant with web and mobile clients.

4.2.2. Gateway variant

The gateway variant can be achieved with patterns like API gateway and backends for frontends, where there are aggregator microservices. If we use the API gateway, there will be one aggregator service that contains client-specific APIs, for example, one for a web application and one for a mobile. This service communicates with each of the needed machine learning microservice, so it can

access models and predictions too. The backends for frontends pattern defines separate API gateways for different kinds of clients; thus, we will have a gateway for web clients and another one for mobile clients, each of them accessing the needed machine learning microservices. In paper [77], where a unified API gateway for high availability clusters was discussed, the authors propose a centralized solution for managing clusters. Our work uses a similar idea, but instead of clusters, we manage other services. We consider the aggregator service to be a microservice, despite its size and complexity. It is responsible for managing data, training, persisting, and accessing models, as well as making ensemble models. The combination of trained models remains on the server side in this variant, as we can have a separate microservice for this purpose. In this architecture, the models are only reachable through an aggregator service, which comes with a downside, in that it will encounter more network load than the other services. To avoid this problem, we could make the aggregator service scalable with multiple copies of it behind a load balancer. Integrating this solution into a backend system is not a complex task, as there is an aggregator service, so each machine learning service can be accessed from it. The downside of this architecture is that both the ensemble service and the API gateway could increase coupling, which depends on the implementation. We suggest this variant for applications that are using many machine learning services, as developers have to use only one connection point, thus the code will be more maintainable. This variant can be seen in Figure 4.2. The blue arrows indicate the path of the data; the green ones are for service registration, but here the mobile and the web application uses the aggregator service to access the needed models' results.



4.2 Gateway architecture variant with web and mobile clients.

4.3. Messages between web services

The whole communication in this microservice application uses the REST API. This means that the web services have to meet the criteria of the REST architectural style. It has a uniform interface as we uniquely identify the resources and send only the necessary data in JSON, and they can be manipulated through representations. When we have two microservices that communicate with each other, one can be identified as a server and the other as the client. Although we do train models and persist the state of them, the application is stateless and only the resources have states. The responses are cacheable as the client can reuse them later for the same requests. The last constraint of REST is the layered system, and in this application, and especially in microservice architecture, the service does not know if it is an intermediary server or not. Almost every service relies on other ones, so at the subsystem level, we could see the layering.

In this system, every workflow involves HTTP requests and responses. We send messages to persist data in the database, train a model with parameters, predict with the given data, or query a model. The goal of this system is to make ML available on web capable devices, so when we have trained models, there are two ways to use them.

The client can request the model itself, which can be used on the client side. This way has downsides, like the size of these models can be really huge,

and sending them through the network can be challenging. In addition, the client must use the same environment as the server does, so a Java object that contains an ML model cannot be used in a Python environment. If we can use the same environment, the advantage of this solution is reusability, as when we successfully transfer a model to the client, it can be used without a network connection, and it will own a local copy of the model.

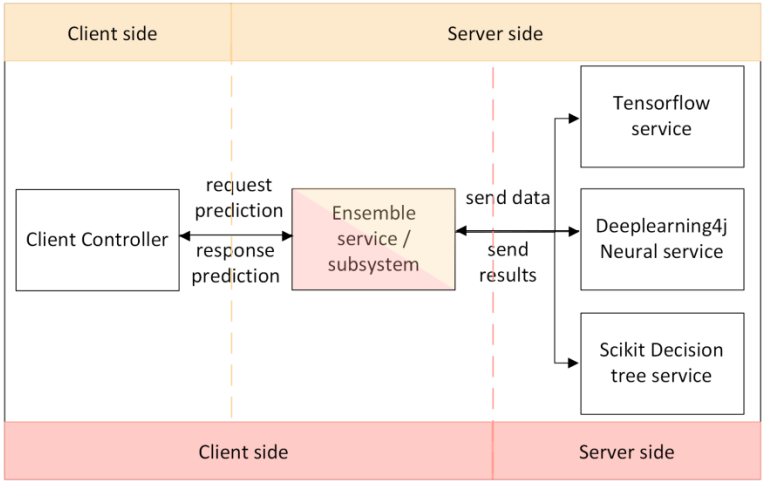
The other way is when the client sends data to the server, which will send back the model's results. This solution is the most commonly used one as the size of messages is usually smaller and they can easily be processed. For example, the client sends the items in a shopping cart, and the response contains a set of items with the probabilities of the customer buying them. We do not have to use the same environment on the client and the server side, as the client does not receive model objects; instead, they get data that is not that complex like a set of numbers or a matrix or a list of strings, which can be processed on a wide range of environments. The only downside is that this solution depends on the network, as for each prediction it must send an HTTP request and process a response, so it must be implemented with caution for devices that do not always have a stable network connection. On devices like mobiles, this problem can be bypassed with a hybrid solution, such as storing a model on the device but updating it from web services. We can use TensorFlow and TensorFlow Lite in this case. In Table 4.1. we can see the endpoints of some microservices in the system.

4.1. Table - Endpoints of the gateway and direct variants

Service	Endpoint	Data	Description
dataup	POST /upload	list	Persist data in the database.
dtree	POST /train	dataset, parameters	Train a decision tree with the given parameters.
dtree	GET /model	model id	Get the trained model from the service.
neural	POST /predict	unlabelled data	Predict the label of an unlabeled data.
dataman	GET /dataset	dataset name	Query for the dataset as a list.
ensemble	GET /model	model type, id	Forward the request to the appropriate microservice.
ensemble	POST /ensemble	method, list of model type, id	Build the given ensemble model using models from other microservices.
ensemble	POST /upload	list	Forward the dataset to the data manager service.

4.4. Combining models of different platforms

The main goal of this application is not just to make machine learning usable as a subsystem, but to allow us to combine different models from different platforms with ensembles. Software developers like to use multiple platforms, languages, web services, and cloud together, and these architectures below allow them to integrate heterogeneous machine learning services with aggregator services. In Figure 4.3, we can see that there is not much difference between the gateway and direct variant when it comes to model combination. The used color codes are the following: the redlike peach pink means direct, and yellowish pastel peach means the other one. It can be seen that the only difference between them is the place of the ensemble service or subsystem. When we do not have an aggregator service, the client has to make ensembles, but if we have one, it can be done on the server side.



4.3 Model ensembling process comparison in the two variants

4.4.1. Direct variant

As there is no API gateway or aggregator service, this variant lacks the ability to combine models on the server side. The client can do this work with an ensemble service or a subsystem. In practice, it looks like the client needs a prediction that should come from multiple trained models on the server side. The first step is to use the appropriate ensemble technique, so it could choose a bagging, boosting, or a simple voting. When we go for voting, the client side sends HTTP requests to the server's model endpoints to get their predictions. For example, it sends data to TensorFlow, decision tree, neural network, and a random forest model, which are written in Python, Java, Python with Scikit-learn, and the random forest comes from the Azure service. Each of them sends back their results to the client as an HTTP response, which can use the voting technique on them. The direct variant's model combination process can be seen in Figure 4.3.

4.4.2. Gateway variant

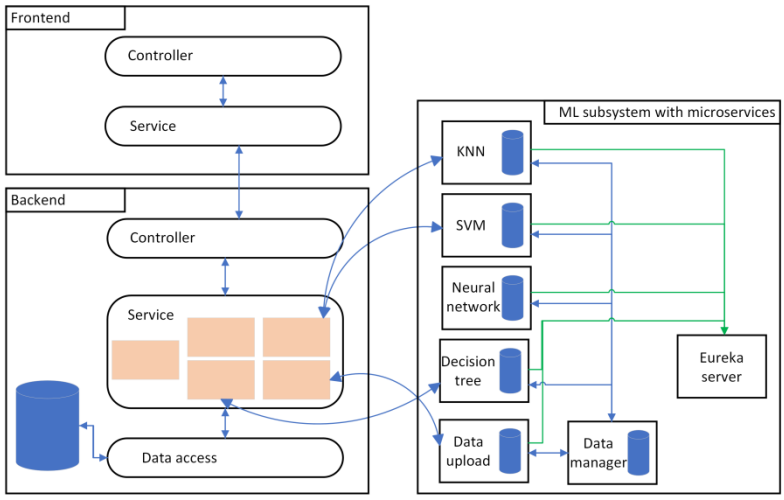
This variant puts the entire computing load on the server side, and thus the combination of models, too. As we have an aggregator service, it can use other model microservices' results in an ensemble model, which can be queried from the client side. When we have different trained models behind services, like a Deeplearning4J and Scikit neural network model, they can be part of the microservice architecture-based application, and their results are accessible from the aggregator or ensemble service. The usual use case looks like the client needs a prediction, and for that, it would use the mentioned Deeplearning4J and Scikit

neural network models. To get their results, it sends an HTTP request to the aggregator service that forwards this request to the model services. The aggregator creates an ensemble model from the responses, and then it can respond back to the client with the results. The gateway variant's model combination process can be seen in Figure 4.3.

4.5. Integration as a subsystem

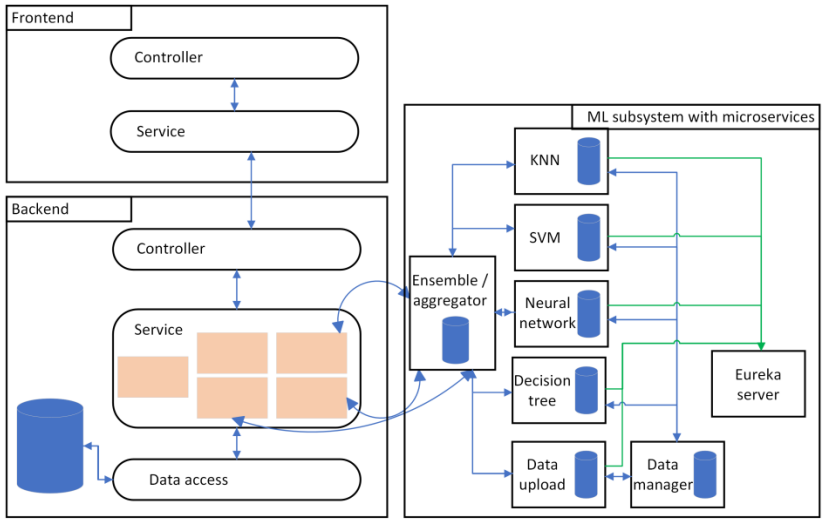
The architectures and concepts above focus on applications that mainly train and manage machine learning models, but it is clear that most of the applications use ML as a subsystem or as a service. To meet this need, the variants detailed can act as a subsystem, so they can be integrated into existing software to improve them with the methods of machine learning. In the integration process, we can distinguish three elements: the clients, which are probably user interfaces, the backend, and the ML variant.

The integration of the direct variant starts with connecting the frontends with the backend software. There is nothing new here; the web or mobile clients query data from the backend and then display it. The connection of ML services to the backend is unique in each application, but it is common that we need to connect backend classes to model endpoints without an aggregator service. This variant is suggested when the main software is smaller or it would use a few machine learning services, because the backend system has to maintain connections with multiple endpoints, and a high number of ML usage would negatively affect software maintainability. There are many advantages of the direct variant in terms of integration. The backend software can minimize network usage by querying only the needed models. Extension can be achieved easily as developers can implement new algorithms and introduce them as new endpoints. Backward compatibility can be maintained as a microservice that can have multiple endpoints for the same data. Without an API gateway, there is no service that processes most of the data, and this could result in a lower risk of critical errors and system shutdowns, because if one endpoint is faulty, only some of the services would become unavailable. The main disadvantage is the difficulties of integration because it requires a lot of programming work on the backend. An implementation can be seen in Figure 4.4.



4.4 Integrating direct variant into a monolithic application.

The integration of the gateway variant is similar to the other; the only difference is in the connection between the backend and the ML subsystem. As there is an API gateway, multiple backend service is connected to it, so they can query the results of machine learning services through it. This can be seen in Figure 4.5.



4.5 Integrating the gateway variant into a monolithic application using the aggregator service.

We suggest the use of this variant when the main software would heavily depend on machine learning services, as integration and maintenance are much

easier than the direct variant, although they share some advantages. For example, both of them are easily extensible and backward compatible. The main risk here is the huge responsibility of the API gateway, because if something goes wrong there, all of the machine learning services would become unavailable.

The developer team can decide which variant is more suitable for their application by the following criteria. How many models would the software use? The more models needed, the more likely the gateway variant is needed. When the application only uses like 2-3 models, it can be faster to use the direct variant as it leaves out the gateway's request/response. How many computing resources does the client have? When the clients are simple computers or mobile devices, their computing capacity is limited, so it is suggested that they use the gateway variant and put the computing load on the server. Who should store the data? If we do not want to store data on client devices, then the gateway variant is more suitable. How much data do we have? Only smaller datasets or dataset slices can be stored on client devices, so direct variant is suggested when we have this kind of data. How many requests will the clients send? Applications can be heavily dependent on services, so it is possible that, for example, a mobile application sends multiple requests per second. The number of request criteria should be interpreted with the number of models. When we need a few models and a lot of requests, the direct variant is preferred, but if we have a few requests too, both variants can be suitable.

The protection of data privacy depends on the implementation of these architecture variants, but this can limit the functionality of them. In the direct variant, ensemble model building is often done on the client side, and because of that, some data must be present on client devices. To avoid this, the client-side tasks should be moved to the server side, and then only the models' result can be sent. The gateway variant can care about data privacy by default, as it can only be accessed through the gateway, and the implementation can restrict what kind of data can go out from the server side and who can access it.

4.5.1. Reference implementation

We made applications for each variant to experience the work with these proposed architectures. Both variants used the same environment, an Intel Core i7-7700HQ processor with 16GB RAM and Windows 10 operating system. The service management software was a Java and Spring-Boot project on a Tomcat application server, and the machine learning part uses Python, Scikit-learn, and Flask. In both implementations, data upload and data management endpoints were part of the Spring application. Although there are exchange formats we could use for sending machine learning models, like ONNX [18], we chose not to use it, because with it, we would have to use a supported framework and

model to make our machine learning solution interoperable. We thought that architecture should not have restrictions like that.

As the field of use is different for each variant, their performance should be measured differently, because comparing them with the same parameters can be misleading. To prove that direct access can be somewhat faster, we made a performance test on both direct and gateway variants. This shows how many requests the endpoint can serve within 1 minute, what is the average response time and the error rate. The direct variant could handle 1085 requests from 20 virtual users with an average of 13.71 requests per second. The average response time is 17 ms, the minimum is 9 ms, and the maximum is 317 ms. The 90% of the requests hit the endpoint within 24 ms. The error rate was 0. As the gateway variant means one extra HTTP request/response pair for each request, its results are indeed a bit worse. From 20 virtual users, it could serve 1004 requests with an average of 12.96 requests per second. The average response time is 51 ms, the minimum is 18 ms, and the maximum is 1221 ms. The 90% of the requests hit the endpoint within 121 ms. The error rate was 0. These results are predictable, as one extra hop and one extra JSON packaging will always have some cost. Although the results of the gateway variant are worse in this scenario, it does not mean that it is less usable in certain environments.

4.6. Possible improvements and conclusion

Training machine learning models and making them more accessible to another system with microservice architecture can be very useful in development, and this chapter presented a unique way of connecting bigger software systems. Making machine learning models portable can be difficult, but hiding them behind scalable web services is easier to implement for many developers, and that is why we avoided the use of ONNX [18] and other formats. These architectures have numerous opportunities for improvement, like comparing them in real applications, examining the scalability, integrating distributed model training or combining them with different kinds of software architectures. The microservices let us extend these web services to perform model or data parallel distributed training, for example training a neural network, where each perceptron is a microservice.

4.7. Relevant thesis

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.

Chapter 5

Novel Directions in ML Application: Number Cognition and Learning Motivation

While the previous chapters focused on the architectural and technical aspects of deploying machine learning in distributed, service-oriented environments, this chapter highlights two distinct, exploratory applications that extend the impact of machine learning beyond infrastructure. These case studies illustrate how machine learning can serve not only as a backend tool but also as a means of advancing cognitive understanding and fostering engagement in educational contexts.

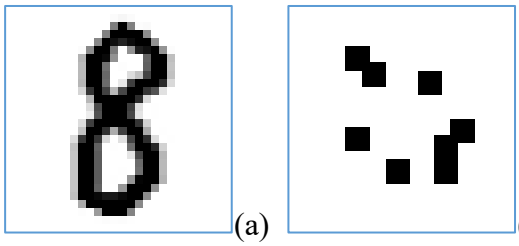
The first section presents a cognitive experiment centered around number cognition, demonstrating how machine learning can be used to investigate the emergence of numerical concepts in artificial systems. The second section introduces a competitive educational framework designed to cultivate interest in machine learning among students, showcasing how carefully designed learning environments can stimulate both motivation and competence in the field.

Together, these examples reveal novel directions in which machine learning can contribute — not only to system design but also to learning, understanding, and inspiration.

5.1. Subitizing in Humans and Machines: Investigating the Capacity of the Object File System with SMNIST Experiments

This chapter is based on a publication in which I was a co-author and entitled Object file system software experiments about the notion of number in humans and machines [O4]. My tasks in the paper of this chapter were to implement a convolutional neural network and train it on the generated images. For reference, I used the famous LeNet model from Yann LeCun. The implementation was made in Java, Deeplearning4J environment. I was involved in the SMNIST for Humans project too, and contributed to the writing of the paper. The idea of this research came from Norbert Bátfai; thus, he was the first author of the paper. Most of the concepts are his work, including the SMNIST datasets. The main author of the cognitive science part is Dávid Papp. The other authors, including me, were developers of the machine learning software. I chose to retain all authors' work in the thesis in order to make my contribution easier to understand.

In the movie Rain Man, Dustin Hoffman's character, Raymond Babbitt, instantly recognizes the exact number of toothpicks on the floor. This inspired us to design a machine learning example, though instead of toothpicks we use dot patterns. The setup parallels the well-known MNIST digit recognition problem [78][79], where classifiers identify handwritten digits such as the "8" shown in Figure 5.1a. In contrast, the "SMNIST for Machines" experiments replace digits with images containing fewer than ten dots (see Figure 5.1b for an example with eight dots). Building on this idea, our research focuses on examining subitizing—the rapid estimation of small numerosities [80].



(a) (b)

5.1 Two typical input images for MNIST (a) and SMNIST (b)

5.1.1.1. Cognitive neuropsychological and computer science background

Research into the biological and psychological roots of numerical ability dates back to the 1930s. Early studies showed, for example, that honeybees can recognize up to four landmarks when seeking food [81]. In neuropsychology, two main frameworks are discussed: (1) the Object File System (OFS), linked to concepts like subitizing, object tracking, and parallel individuation; and (2) the Analogue Number System (ANS), also known as the Approximate or Analog Magnitude System [82][83][84][85][86][87][88]. The OFS supports "numerosity," the capacity to estimate the number of objects in a set without counting—accurately up to about four items in humans [85][86]. Thus, the OFS helps identify small quantities (maximum four) by assigning distinct markers to each object [82].

Studies on vertebrates such as cats [89][90], chimpanzees [91][92], and parrots [93] have explored the biological and evolutionary basis of numerical ability. Parallel research on infants examined the development of the OFS [94][95][96][97][98][99], showing that numerosity is present early in life [85], or at least emerges naturally within innate systems [100].

The Analogue Number System (ANS) is found in many animals and humans, enabling estimation of small quantities without symbols or language. As

development progresses, its precision improves (reflected in decreasing Weber fractions), making it a cornerstone of numerical cognition [101][102][103][104][105][106]. Cognitive research has used both simulations and mathematical models—for instance, [107] designed a neural architecture to measure distance and size effects [108][109], while [110] modeled the OFS capacity directly. Both OFS and ANS appear to have evolutionary origins [111][112], likely refined over hundreds of millions of years [88], whereas mathematics itself emerged only thousands of years ago. Still, mathematics may also trace back to evolutionary roots, as suggested by Newton's laws [113] and Darwinian neurodynamics [114].

From the viewpoint of computer science, the numerical abilities of computers are of analogue or digital nature [115]. In today's digital computers, numbers are represented in either fixed-point or floating-point format [116]. Obviously, in contrast with previously exemplified neuropsychological systems, the numerical fundamentals of computers are fully known, because they have been developed as a result of targeted research and engineering processes [117]. However, it should be noted that it would not be necessarily true for systems that include some deep learning black box artificial intelligence (AI) elements [118].

In this chapter, we extend prior work on numerosity in vertebrates and infants to machine learning programs. Similar efforts already exist online, showcasing state-of-the-art models based on artificial neural networks (ANNs) such as convolutional neural networks (CNNs) [119][120] and multilayer perceptrons (MLPs). Standard benchmarks like MNIST [79] and CIFAR-10 [121] are also used in our experiments. Today, deep learning often outperforms humans in tasks such as classic video games [122], GO [123], Quake III Capture the Flag [124], and Starcraft [125]. These advances trace back to early foundations: data-flow programming [126], mathematical neuron models [127], and the perceptron [128]. Major AI groups now pursue Artificial General Intelligence (AGI) using platforms such as Microsoft's MALMO Minecraft [129] or Google's DeepMind Lab-Quake III [130][131], both built on well-known games. Notably, Lamarckian evolutionary approaches have also appeared in this field [132][133].

Recently, the VisNumBench benchmark [134] has been used to test whether multimodal large language models (MLLMs) have a true "number sense." Covering over 1,900 visual tasks, it showed that even leading models fall short of human performance, especially beyond the subitizing range. Similarly, Testolin, Hou, and Zorzi examined visual enumeration in generative AI models [135], finding that systems like GPT-4V and Gemini also struggle with accurate numerosity estimation outside the subitizing limit. Together, these studies

confirm that despite advances in vision–language integration, current models still lack a robust, generalizable concept of number.

5.1.1.2. Current study

The experiments are divided into two parts: “SMNIST for Humans” and “SMNIST for Machines.” The human study does not measure numerosity estimation directly [136] but instead identifies the transition between OFS and ANS. Results confirm earlier findings that OFS handles sets of 1–4 items [137], while larger quantities are processed by the ANS.

While “SMNIST for Humans” explores the limits of OFS, “SMNIST for Machines” introduces benchmark datasets to test how deep learning models handle numerosity classification, despite being designed for other tasks. Similar work includes Wu, Zhang, and Shu [138], who studied number representation in machines, though with a simpler subitizing setup. Related studies include an unsupervised simulation replicating macaque results [139][140], though unlike ours it lacked supervised learning and neuron-level analysis. Following Chen et al. [141], we also examined small numerosities near the OFS limit. Other emergentist approaches compare humans and neural nets [142], while Hannagan et al. [143] describe core properties of numerosity estimation, suggesting possible links to the cultural emergence of zero [144].

Some existing software already performs tasks similar to the “SMNIST for Humans” benchmark, such as measuring ANS performance and calculating the Weber fraction. In this study, however, the focus is on OFS rather than ANS. A broader, more aspirational goal is to develop a program capable of simulating the cognitive evolution of numbers and forming a notion of numerosity itself [145].

5.1.2. EXPERIMENT 1. SMNIST for Humans

The term “SMNIST for Humans” comes from the “SMNIST for Machines” experiment (Experiment 2). Its only link to classical MNIST is structural: both classify inputs—digits or numerosities—into 10 classes.

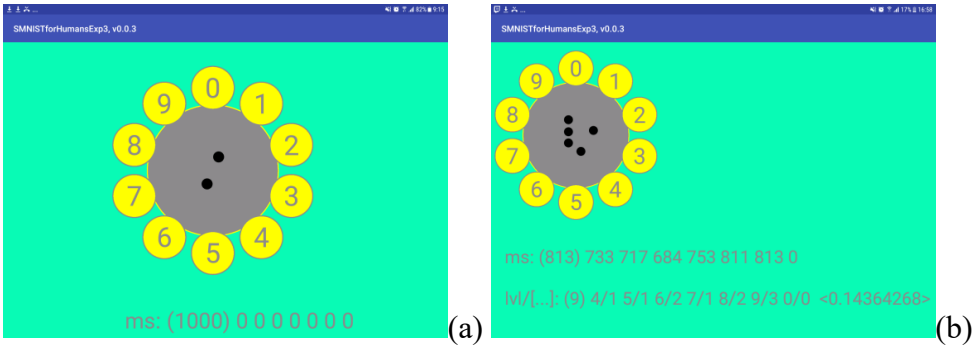
5.1.2.1. Method

The “SMNIST for Humans” Android app serves as a benchmark for testing the human parallel individuation system. It is available in the GitLab project [146] (Humans/SMNISTforHumansExp3) and can also be downloaded as an APK from the SMNIST dataset website[1].

[1] http://smartcity.inf.unideb.hu/~norbi/SMNIST/SMNISTforHUMANS/ Exp3/ Original souce unavailable. Alternative:

As shown in Figure 5.2a, the app displays a set of dots, and the user must select the correct digit within a limited time. In this experiment, dots are circles of 101-pixel radius, while other versions use smaller 1x1 or 3x3 pixel patterns (squares, outlines, X, O, or +) depending on the dataset. The time window starts at 1000 ms but is dynamically adjusted for smoother gameplay. Figure 5.2b shows level changes and average numerosities displayed alongside timing values. Players can resize dots with Android gestures, and new dot patterns appear about every 700 ms to discourage counting.

Players begin at level 3, where 0–2 dots appear randomly. Advancing requires 10 consecutive correct answers, after which the next level is unlocked. Figure 5.2b (second row) shows the achieved levels and the mean numerosities during transitions. For example, moving from level 3 to 4, the average of ten successful trials might equal 1 (e.g., sequence 0,0,2,1,2,2,0,1,2,1). The notation continues similarly for higher levels, while 0/0 indicates level 10 is not yet reached. A heuristic score, $\sum_{i=3}^{level} \frac{(l_i+1)(i+1)}{s_i}$, where l_i is the mean numerosity and s_i the time in ms, is also calculated (first row in Figure 5.2b). This value is included as a gamification element, with higher scores reflecting better performance

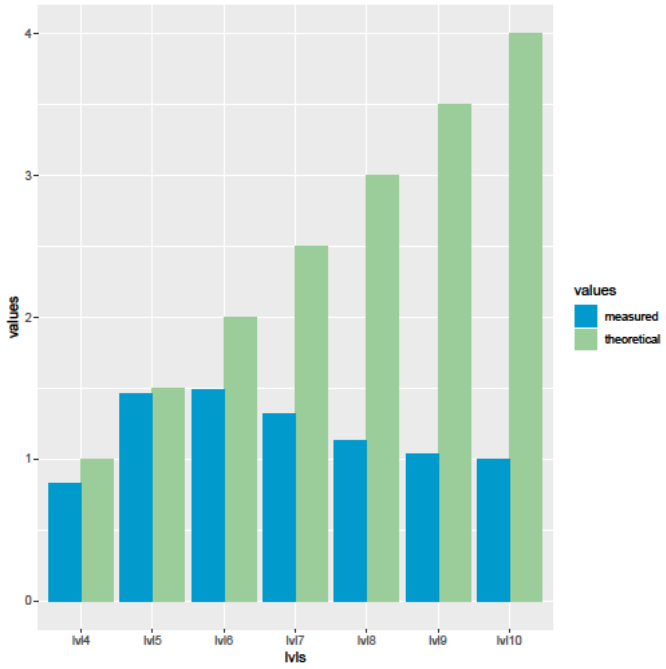


5.2 SMNIST for Humans. (a) Rapid prototype. (b) Experiment 3”

Data were collected via a closed Facebook group (UDPROG) of 680+ current and former Computer Science BSc students at the University of Debrecen. Most participants were second-semester students, predominantly male, with only six female participants. Completing the test allowed students to replace a homework task. In total, 104 Android screenshots were submitted, one

https://web.archive.org/web/20220709085940/http://smartcity.inf.unideb.hu/~norbi/SMNIST/SMNISTforHUMANS/Exp3/

shown in Figure 5.2b. Although the screenshots included reaction times, these were not analyzed in this experiment.



5.3 Theoretical and measured mean of the number of dots.

Figure 5.3 presents the measurement results, while Table 5.1 provides detailed and derived data. Theoretical and mean values align, with SD = standard deviation, CV = coefficient of variation [136], and w = internal Weber fraction [147]. Unlike Panamath, which compares two numerosities, our setup shows only one, so level transitions (e.g., 3→4, 4→5, … 9→10) are treated as comparisons. Based on results, w ≈ 0.25. These findings agree with cognitive psychology studies [83][84][86][87], confirming that the parallel individuation system handles fewer than four items. This is demonstrated in Figure 5.3, where the measured average of the integer parts of means of randomly picked (and of course consecutive successfully detected) 10 integers is lagging far behind the theoretical expected value of the mean of randomly picked 10 integers from 0 to level-2, inclusive, that it simply grows linearly with level, namely it is equal to (level − 2)/2 (where level starts from 4), because we use uniform distribution that is $r_i \in [0, \text{level}-2]$ $(\sum_{i=1}^{10} r_i/10) = \sum_{i=1}^{10} E(r_i)/10 = \sum_{i=1}^{10} (level-2)/20$. In this interpretation of levels, level n denotes the event of changing level from n-1 to n. For example, level 4 means that 0, 1, or 2 dots have already been

successfully detected (and now the 0, 1, 2, and 3 values are being selected randomly).

5.1. Table - Detailed and derived data of Figure 5.3.

	lvl4	lvl5	lvl6	lvl7	lvl8	lvl9	lvl10
theoretical value	1	1,5	2	2,5	3	3,5	4
measured value	0,826923	1,461538	1,490385	1,314286	1,133333	1,036364	1
mean	0,826923	1,461538	1,490385	1,314286	1,133333	1,036364	1
SD	0,544947	0,919062	1,117993	1,177684	1,024153	1,043833	1,195229
CV	0,659006	0,628832	0,750137	0,896064	0,903664	1,007207	1,195229
w	0,333333	0,25	0,2	0,166667	0,142857	0,125	0,111111
CV/w	1,977018	2,515327	3,750685	5,376382	6,325649	8,057656	10,75706

A possible improvement of “SMNIST for Humans” is refining dot separation to prevent near-overlaps, since in the current version dots can appear only one pixel apart, potentially distorting results [148]. Future extensions toward measuring ANS (with numerosities beyond 0–9) must also account for visual features of the stimuli [149]. This is particularly relevant here, as custom-sized dots were used, as shown in Figure 5.2b.

5.1.3. Deeplearn4j LeNet MNIST modifications

The aim of the modified DL4J example is to train and evaluate a classical LeNet-style convolutional neural network (CNN) on the SMNIST datasets. This program is a fork of the canonical examples in the DL4J repository. In the case of CNNs, it is essential to specify InputType.convolutionalFlat(...), which ensures that the 28×28 flattened vectors are automatically reshaped by the framework into a 4D tensor (minibatch, channel, height, width). The DL4J developers explicitly highlight this requirement.

5.1.3.1. Data loading and preprocessing

The example typically employs the MnistDataSetIterator to read images and labels in minibatches (commonly of size 64 or 128). The iterator normalizes pixel values to the [0,1] range and produces one-hot encoded labels for the 10 classes (digits 0–9).

5.1.3.2. Input format

InputType.convolutionalFlat(28,28,1) converts the flattened 784-dimensional vectors into the DL4J CNN-compatible [NCHW] format. It is important not to configure this manually via nIn; for convolutional networks in DL4J the proper approach is always to use InputType.

5.1.3.3. Network architecture (LeNet-style)

Although layer parameters may vary slightly across versions, the typical architecture follows the sequence: Conv2D → Subsampling(2×2) → Conv2D → Subsampling(2×2) → Dense → Output (Softmax, 10 classes).

The activation function is ReLU (earlier variants occasionally use alternatives for teaching purposes). Weight initialization typically follows Xavier/He initialization. The output layer applies Negative Log Likelihood combined with Softmax across the 10 classes. This configuration reflects the canonical LeNet design, as also summarized in the DL4J "Model Zoo" descriptions.

5.1.3.4. Training configuration

Optimization is carried out with Nesterov momentum (SGD with momentum ≈0.9) and a learning rate usually in the range of 1e-2 to 1e-3. Regularization is provided by an L2 penalty (≈1e-4). Training is run for several epochs, often monitored using the ScoreIterationListener. Evaluation is based on accuracy and confusion matrices computed on the MNIST test set.

5.1.4. EXPERIMENT 2a. SMNIST for Machines

This chapter does not attempt to build new numerosity-recognition models but instead tests well-known models designed for other purposes to see if they show such abilities. This approach aligns with the emergentist perspective [100]. From this viewpoint, using MNIST as inspiration was intuitive, and our experiments can be seen as a semantic variant of the classical MNIST.

5.1.4.1. Method

"SMNIST for Machines" was designed as a standardized task to test a program's ability to recognize dot numerosities. Unlike "SMNIST for Humans," which needs only test data generated during gameplay, this version requires both training and test datasets.

5.1.4.2. The Generator Program

The SMNIST datasets were created with our own generator, producing images of fewer than 10 dots. The output is fully binary compatible with the

original MNIST format [79], allowing researchers to reuse MNIST-based programs directly for SMNIST experiments.

5.1.4.3. SMNIST datasets.

The datasets are divided into two releases ("SMNIST for Machines" and "SMNIST for Anyone"), each with two development series. In the first series of "SMNIST for Machines," six train/test pairs of 28×28 images were created.

- Naive: this set includes 60,000 training and 10,000 test images, each with fewer than 10 randomly placed, centered dots (represented as 3×3 pixel squares[2] due to the small image size). Table 5.2 shows the histogram of this dataset. A chi-square test confirmed the distribution is consistent with uniformity.

5.2. Table - The histogram of the dataset "SMNIST for Machine" Naive.

	0	1	2	3	4	5	6	7	8	9
train	6025	5977	5965	5928	6075	6067	6004	5930	6051	5978
test	986	1008	980	963	1064	970	996	1010	1036	987

- No-centering: Generated like the Naive set, but dots are not centered.
- Disjunct: All images are unique, so training and test sets do not overlap (except for the single case of 0 dots, which appears in both). Table 5.3 shows the histogram of this dataset.

5.3. Table - The histogram of the dataset "SMNIST for Machine" Disjunct.

	0	1	2	3	4	5	6	7	8	9
train	436	436	7390	7166	7482	7491	7299	7352	7498	7450
test	49	48	1215	1213	1227	1263	1282	1210	1209	1284

- Disjunct 1PX: Similar to previous sets, but dots are reduced from 3×3 squares to single 1×1 pixels.
- Hard: Pixel coordinates are split into two disjoint sets, with training images drawn from one and test images from the other. Table 5.4 shows the histogram, where the 22×22 = 484 pixels are divided into subsets of 425 and 59.

[2] https://gitlab.com/nbatfai/smnist/blob/master/smnistg/smnistg.cpp#L158)

5.4. Table - The histogram of the dataset “SMNIST for Machines” Hard

	0	1	2	3	4	5	6	7	8	9
train	6751	425	6651	6656	6531	6646	6678	6482	6715	6465
test	1107	59	1146	1089	1045	1101	1113	1118	1141	1081

- Hard1PX: Generated like the Hard set, but with 1×1 pixel dots.

The second series contains training and test images only of size 10x10 pixels with dots of 1x1 pixel described precisely by the following:

- Disjunct: As before, train and test sets are disjoint, but here exactly one no-dot image exists in each. Table 5.5 shows the histogram of this dataset.

5.5. Table - The histogram of the dataset “SMNIST for Machines” 10x10 Disjunct

	0	1	2	3	4	5	6	7	8	9
train	1	90	4455	7926	7806	8008	7940	8069	7872	7833
test	1	10	438	1382	1315	1352	1347	1441	1379	1335

- Hard: Similar to the previous case, but with 0-dot images handled separately. Table 5.6 shows the histogram, where the 10×10 = 100 pixels are split into two disjoint sets of 84 and 16.

5.6. Table - The histogram of the dataset “SMNIST for Machines” 10x10 Hard

	0	1	2	3	4	5	6	7	8	9
train	1	84	3486	8126	7943	8034	8061	8115	8003	8147
test	1	16	120	560	1567	1571	1518	1501	1534	1612

- Disjunct pow 102x+, Hard pow 102x+: Similar to earlier sets but with a different probability distribution for generating n ∈ {1,...,m} dots in train images is the following: $F(x) = \begin{cases} 0 \; for \; x \leq 1 \\ \frac{10^{2x}}{10^{2m}} for \; 1 < x \leq m \\ 1 \; for \; x \geq m \end{cases}$ where m denotes the maximum digit. In cases of these training and test sets, m is equal to 9. The reason for choosing this distribution is that there are 100!/(100−n)! possible 10x10 images that contain exactly n (different, order matters) dots (variations without repetition). To be more precise, in our case it is equal to $\binom{100}{n}$ because all n pixels are the same color (order

does not matter, combinations without repetition). In practice, histogram soft-generated images follow the case of combination without repetition due to the uniqueness condition of the Disjunct (and all further) datasets. It should be noticed that the number of dots in the test images still follows uniform distribution. In addition, due to using the above distribution that produces histograms, for example, like this 0: 1, 7: 5, 8: 600, 9: 59394 where the numbers of dots 2, 3, 4, 5, and 6 are missing, therefore the ten percent of generation of training images follows uniform distribution. In the case labelled by "Disjunctpow102x+", we can see the histograms in 5.7 Table

5.7. Table - The histogram of the dataset "SMNIST for Machines" Disjunct pow 102x+

	0	1	2	3	4	5	6	7	8	9
train	1	71	719	717	720	770	763	792	1302	54145
test	1	29	1246	1234	1222	1275	1260	1248	1204	1281

In the other ("Hard pow 102x+") case, the histograms can be seen in 5.8 Table, where the 10*10=100 pixels are divided into two disjoint sets of sizes 84 and 16.

5.8. Table - The histogram of the dataset "SMNIST for Machines" Hard pow 102x+

	0	1	2	3	4	5	6	7	8	9
train	1	84	732	751	711	724	773	766	1199	54259
test	1	16	120	560	1525	1617	1523	1574	1512	1552

- These sets follow the same scheme as *Hard pow 102x+* but with m = 4, 5, 6, 7, or 8. Table 5.9 shows the histogram for "4H-102x+." Here, the 10×10 = 100 pixels are split into subsets of 72 and 28. The "theoretical" column lists the possible image counts for n = 0–4 dots, while the "statistics" column shows the generated images.

All the data used in this experiment can be found at the SMNIST dataset website[3]. The same data can also be found at GitLab [146] under the directory Datasets/SMNIST.

[3] http://smartcity.inf.unideb.hu/~norbi/SMNIST/ Source currently unavailable. Alternative:

5.9. Table - The histogram of the dataset “SMNIST for Humans” Series 2/4H-102x+.

72/28	theoretical		statistics	
dots	train	test	train	test
0	1	1	1	1
1	72	28	72	28
2	2556	378	1925	378
3	59640	3276	2574	3276
4	1.02879e+06	20475	55428	6317

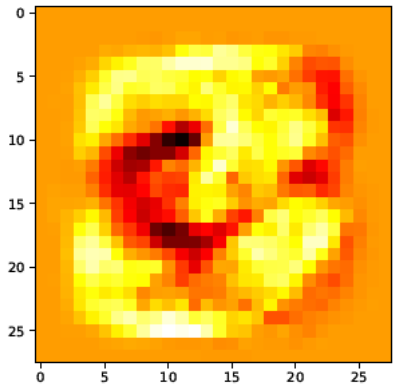
5.1.4.4. Running results

For measurements, we have used the following well-known programs and models with default or different settings and with minor modifications in some certain cases:

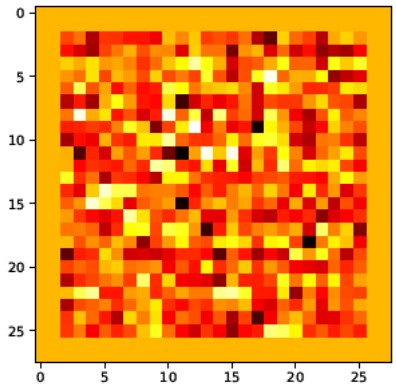
1. Tensorflow 0.9.0, mnist_softmax.py[4] (softmax regression) [48][150]
2. Tensorflow 0.9.0, mnist_softmax.py UDPROG is the same as the previous one, but it contains extensions for printing out debug messages (for example, it draws the well-known visualizations of MNIST tutorials5 shown in Figure 5.4, Figure 5.5, and Figure 5.6);

https://web.archive.org/web/20220709012938/http://smartcity.inf.unideb.hu/~norbi/SMNIST/

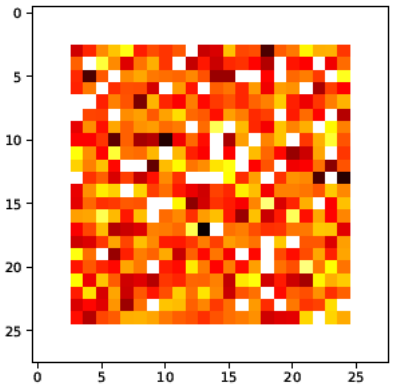
[4] https://github.com/tensorflow/tensorflow/releases/tag/v0.9.0 (running with TF version 1.13.1)



5.4 Typical weights for classification of 3.



5.5 Weights for classification of 3 using the SMNIST Series 1/NoCtrg dataset.



5.6 Weights for classification of 3 using the SMNIST Series 1/H-1PX dataset.

3. Tensorflow 1.4, mnist_deep.py[5] (convnet)
4. Keras mnist_cnn.py[6] (convnet) [151]
5. PyTorch cifar10_tutorial.py[7] (convnet) [152]
6. deeplearning4j, based on the LeNet [78] MNIST example[8] [153]
7. Swift TF 2-layer MLP with softmax[9], partially based on Tensorflow swift-models[10] [154]

---

[5] https://github.com/tensorflow/tensorflow/blob/r1.4/tensorflow/examples/tutorials/mnist/mnist_deep.py (running with TF version 1.13.1)

[6] https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py (running with TF version 1.13.1) Original source unavailable. Alternative: https://github.com/keras-team/keras/blob/7a39b6c62d43c25472b2c2476bd2a8983ae4f682/examples/mnist_cnn.py

[7] https://github.com/pytorch/tutorials/blob/master/beginner_source/blitz/cifar10_tutorial.py

[8] https://github.com/deeplearning4j/deeplearning4j-examples/blob/7b57aa8aff5b76cea7944f940c145be1bc75c54e/dl4j-examples/src/main/java/org/deeplearning4j/examples/convolution/LenetMnistExample.java

[9] https://colab.research.google.com/drive/1NYzgkQAc8OZHVrr-6GOVFaYVT7WjW574

[10] https://github.com/tensorflow/swift-models

8. Swift TF MNIST[11] (convnet)
9. Swift TF CIFAR Keras[12] (convnet)
10. Swift TF CIFAR PyTorch[13] (convnet),
11. Keras/ Hierarchical RNN mnist_hierarchical_rnn.py[14] [155][156]
12. MXNet 1.2.1[15] [157], based on a CNN MNIST example[16] and Apache MXNet[17]
13. Lasagne[18] (convnet) [158].

[11] https://github.com/tensorflow/swift-models/blob/master/MNIST Original source unavailable. Alternative: https://github.com/tensorflow/swift-models/tree/01ff1f457a827ce0fd4698866dbba69c94a889f5/MNIST

[12] https://github.com/tensorflow/swift-models/blob/master/CIFAR/Models.swift Original source unavailable. Alternative: https://github.com/tensorflow/swift-models/tree/tensorflow-0.4/CIFAR/Models.swift

[13] https://github.com/tensorflow/swiftmodels/blob/master/CIFAR/Models.swift; Original source unavailable. Alternative: https://github.com/tensorflow/swift-models/tree/tensorflow-0.4/CIFAR/Models.swift

[14] https://github.com/kerasteam/keras/blob/master/examples/mnist_hierarchical_rnn.py Original source unavailable. Alternative: https://github.com/keras-team/keras/blob/tf-keras/examples/mnist_hierarchical_rnn.py

[15] https://github.com/kovacsferencz98/SMNIST_proto/blob/master/smnist_mxnet.py

[16] https://www.tensorflow.org/tutorials/estimators/cnn Original source unavailable. Alternative: https://web.archive.org/web/20190130121124/https://www.tensorflow.org/tutorials/estimators/cnn

[17] https://mxnet.incubator.apache.org/versions/master/tutorials/python/mnist.html. Original source unavailable. Alternative: https://web.archive.org/web/20190522182009/https://mxnet.incubator.apache.org/versions/master/tutorials/python/mnist.html

[18] https://github.com/Lasagne/Lasagne/blob/master/examples/mnist.py

5.10. Table - Measurements with “SMNIST for Machines” Series 1

Program	MNIST	Naive	No-Ctrg	Disjunct	D-1PX	Hard	H-1PX
Tensorflow 0.9.0, mnist_softmax.py	0.9166	0.6078	0.6233	0.5616	0.3888	0.5779	0.1107
Tensorflow 0.9.0, mnist_softmax.py, UDPROG	0.9187	0.6249	0.6072	0.5959	0.4397	0.6025	0.1107
Tensorflow1.4, mnist_deep.py	0.9925	0.9787	0.9558	0.9608	0.9903	0.9592	0.9941
Keras 2.2.4, mnist_cnn.py	0.9908	0.9415	0.9268	0.9446	0.9997	0.911	0.9997
Keras/Hierarchical RNN	0.9858	0.965	0.9828	0.9754	0.9974	0.9386	0.9655
PyTorch, cifar10_tutorial.py	0.9907	1.0	0.9932	0.8973	0.9957	0.8661	0.88
deeplearning4j LeNet MNIST	0.9848	0.9929	0.9842	0.9638	0.9886	0.9496	0.9957
MXNet1.2.1,smnist_mxnet.py	0.991	0.9717	0.9763	0.9436	0.9842	0.8911	0.9843
Lasagne, mnist.py	0.9924	0.9362	0.9238	0.9235	0.9874	0.8970	0.9856

5.1.4.5. Results with SMNIST for Machines Series 1

It is quite obvious that all the programs produce good performance on the original MNIST dataset, as it can be seen in the first column of Table 5.10. The running results for Series 1 of our datasets are shown in further columns. The first two rows show that softmax regression models do not perform well, but this is not surprising if we take a look at Figure 5.4, Figure 5.5, and Figure 5.6, where the visualizations of weights for the classification of digit 3 can be compared. Figure 5.4 shows the well-known typical weights for classification of 3 in Tensorflow 0.9.0, mnist_softmax.py (UDPROG) using the classical MNIST dataset. It may be noticed that the positive weight values draw out the silhouette of the digit 3. Figure 5.5 shows the weights for classification of 3 in Tensorflow 0.9.0, mnist_softmax.py (UDPROG) using the SMNIST Series 1/NoCtrg dataset. The images of Series 1 datasets have a rectangular border of some pixels because the coordinates of dots are generated from range [4, 24]. Figure 5.6 shows the weights for classification of 3 in Tensorflow 0.9.0, mnist_softmax.py (UDPROG) using the SMNIST Series 1/H-1PX dataset. The 59 test pixels (and

the 300 pixels of the border) are white. In contrast, the more sophisticated models like the deep CNNs perform on the "SMNIST for Machines" dataset significantly better than the softmax regression.

As an exploratory step, the PyTorch model was adapted into a DQN model [159] for comparison. This ensured that performance differences came only from the learning approaches (supervised vs. reinforcement learning). A custom environment was created where the model classified images from the Series1/Naive dataset; if accuracy exceeded a threshold, play continued with the threshold raised, otherwise the episode ended. Although accuracy improved slightly with training, the DQN results remained poor, often around 0.4, 0.3, or 0.2. Adjusting episodes, sampling, or hyperparameters (gamma, epsilon, memory size, etc.) did not significantly improve performance.

5.11. Table - Measurements with "SMNIST for Machines" Series 2

Program	Disjunct	Hard	D-102x+	H-102x+
Tensorflow 0.9.0, mnist_softmax.py, UDPROG	0.6066	0.056	0.1281	0.1512
Keras 2.2.4, mnist_cnn.py	0.8822	0.7648	0.8145	0.4625
Keras/Hierarchical RNN	0.9995	0.9999	0.9965	0.9897
PyTorch, cifar10_tutorial.py	0.9528	0.6243	0.8776	0.5365
deeplearning4j LeNet MNIST	0.8488	0.4895	0.2757	0.2388
MXNet1.2.1,smnist_mxnet.py	0.653	0.4013	0.3668	0.3005

5.1.4.6. Results with SMNIST for Machines Series 2

In this series, image size was reduced from 28×28 to 10×10 pixels, and the training distribution was adjusted so that the probability of generating n dots reflected the number of possible placements. Details are given in the dataset descriptions. Most models showed weaker performance on H-102x+ (last column of Table 5.11), leading to its division into subsets 4H-102x+ through 8H-102x+ (Table 5.12). Based on the "SMNIST for Humans" results, we expected performance to drop as the number of dots increased. This pattern appeared in

most models, though exceptions exist, such as Keras/HierarchicalRNN. Overall, the programs performed well when numerosities stayed within the human OFS capacity, i.e., below about four. A targeted test with smaller numerosities (H3-102x+) is presented in the next section.

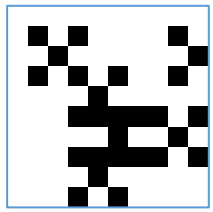
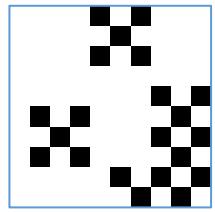
5.12. Table - Measurements with "SMNIST for Machines" Series 2 with particular attention to the further breakdown of the set Hard pow 102x (H-102x+)

Program	4H-102x+	5H-102x+	6H-102x+	7H-102x+	8H-102x+
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 Py-Torch	0.6729	0.6218	0.4796	0.4156	0.4802

5.1.5. EXPERIMENT 2b. SMNIST for Anyone

5.1.5.1. Method

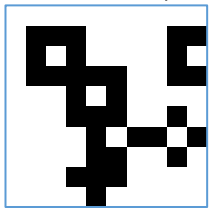
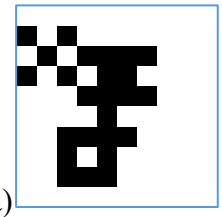
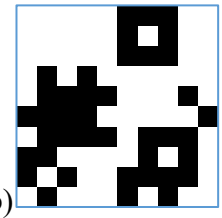
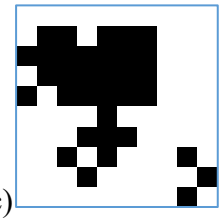
"SMNIST for Anyone" extends "SMNIST for Machines," based on the idea that if machines can solve these tasks, humans should as well. Unlike "SMNIST for Humans," no dedicated test software yet exists, though it would build on the Android app. This experiment goes beyond subitizing, since solutions are not always unique and may involve combinatorial reasoning, allowing multiple correct estimations. The datasets are structured like the 4H-102x+ … 9H-102x+ sets, but dots are replaced by 3×3 pixel patterns of symbols ('X', 'O', '+', and square outline 'S'). Examples are shown in Figures 5.7 and 5.8, both containing six 'X's, though Figure 5.8 demonstrates cases with six or nine mixed symbols. Importantly, the test is not uniquely defined, as in many instances the number of placed objects is ambiguous.


(a)

(b)

5.7 SMNIST for Anyone, Series 1. (a) smnistgtrain-6-7; (b) smnistg-train-6-8

5.1.5.2. Measurement results with SMNIST for Anyone Series 1

The images of Series 1 contain only 3 x 3 pixels binary patterns of 'X's. In all the cases, the performance has already deteriorated with the increasing number of dots, as can be seen in Table 5.13.


(a)

(b)

(c)

(d)

5.8 SMNIST for Anyone, Series 2.

5.13. Table - Measurements with SMNIST for Anyone Series 1

Program	H4-102x+	H5-102x+	H6-102x+	H7-102x+	H8-102x+	H9-102x+
Tensorflow 0.9.0, mnist_softmax.py	0.6317	0.3188	0.2334	0.1942	0.1671	0.1402
Keras 2.2.4, mnist_cnn.py	0.836	0.7546	0.6914	0.6702	0.6233	0.5913
Keras/Hierarchical RNN	0.8498	0.7152	0.6896	0.6537	0.5144	0.5498
deeplearning4j LeNet MNIST	0.6862	0.6764	0.3845	0.3394	0.3008	0.2245

5.1.5.3. Measurement results with SMNIST for Anyone Series 2

In Series 2, images may include the symbols 'X', 'O', '+', and square outline ('S'). As shown in Table 5.14, program performance was consistent with the earlier experiments. A separate test, H3-102x+, limited to a maximum of three objects, is detailed in Table 5.15. As predicted, models performed strongly

here—for instance, Keras 2.2.4, mnist_cnn.py reached 0.9436 accuracy, while Keras/Hierarchical RNN achieved 0.9522.

5.14. Table - Measurements with “SMNIST for Anyone” Series 2

Program	H4-102x+	H5-102x+	H6-102x+	H7-102x+	H8-102x+	H9-102x+
Tensorflow 0.9.0, mnist_softmax.py	0.6317	0.3217	0.2388	0.192	0.1638	0.1399
Keras 2.2.4, mnist_cnn.py	0.7959	0.5627	0.6036	0.5504	0.516	0.4696
Keras/Hierarchical RNN	0.7366	0.595	0.5198	0.5014	0.5099	0.4406
deeplearning4j LeNet MNIST	0.6558	0.4166	0.3566	0.3203	0.2743	0.2304

5.15. Table - The histogram of the dataset Series 2/H3-102x+

57/43	theoretical		statistics	
dots	train	test	train	test
0	1	1	1	1
1	57	43	57	43
2	1596	903	1567	903
3	29260	12341	28375	9053

5.1.6. Conclusion

All experiments in this chapter addressed numerosity, focusing on OFS without direct human–machine comparison. The “SMNIST for Humans” app was designed for people, while “SMNIST for Machines” targeted programs. Our key finding is that the performance shift seen in humans also appears in standard machine learning models. Human results matched cognitive psychology literature, while the “SMNIST for Machines” and “SMNIST for Anyone” experiments suggest that ANNs classify numerosities most accurately when they remain below the human OFS limit that is roughly 4.

5.1.7. Relevant thesis

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.

5.2. Red Flower Hell: a Minecraft MALMÖ Challenge to Support Introductory Programming Courses

This chapter is based on a publication in which I was a co-author and entitled Red Flower Hell: a Minecraft MALMÖ Challenge to Support Introductory Programming Courses [O5]. I was involved in the development of the Red Flower Hell software, where my responsibilities included collecting data and summarizing the results. I also made significant contributions to writing the paper associated with this chapter. The research concept originated from Norbert Bátfai, who taught the High-level Programming Languages course mentioned later. He implemented the initial programs in the MALMÖ environment and was responsible for organizing the competitions. The other authors were students of the course, whose role was to share their experiences and contribute to writing the paper.

The Red Flower Hell challenge was created for undergraduate courses (e.g., programming or AI) to provide a competitive learning experience. Its name reflects the task: agents must collect as many red flowers as possible in a Minecraft gorge designed for the challenge, where lava flows down the slopes in a battle-royale style. In round one, agents only avoid lava; in round two, they also face Minecraft monsters such as zombies and spiders. Agent-vs-agent combat has not yet been implemented. The challenge was built with Microsoft's Project MALMÖ, a Minecraft mod for AGI research [129], and tested in the High-level Programming Languages course at the University of Debrecen in spring 2019/2020. The integration aimed to modernize programming education and make it more engaging for students.

5.2.1. Introduction to High-level Programming Course

In the High-level Programming course at the University of Debrecen, students worked on a "yearbook of programmers," completing tasks and documenting them. At the end of each chapter, a Minecraft MALMÖ challenge was included as a core requirement. Optional tasks provided extra experience points, motivating students to engage further with MALMÖ and leading to ongoing learning and competitions. Early tasks taught basics, such as moving characters or retrieving visual data, while later ones included competitive phases of the Red Flower Hell challenge (see Section 4) and live-streamed events like Red Flower Hell III–IV, where students followed their agents' performance. These activities differed from traditional assignments, which made them highly enjoyable. Overall, MALMÖ-related tasks helped first-year students strengthen programming and Python skills, fostered algorithmic thinking, and offered

immediate visualization of results—useful even for introductory programming courses.

5.2.1.1. Minecraft and AGI Research

The MALMÖ AGI platform continues to develop through competitions with leading universities and companies [160][161][162], and at the University of Debrecen, the DEAC-Hackers esports division also has a Minecraft-MALMÖ team [163]. MALMÖ is Microsoft's only Minecraft mod, created for AI research, and games like Minecraft provide ready-made environments for such studies, simulating real-world elements like hunger, damage, crafting, and problem-solving. This made it suitable for comparing intelligent agents with human intelligence, such as in the Human Intelligence competition tied to the Red Flower Hell task.

Other companies also link gaming and AI, most notably Google DeepMind. Its AI AlphaStar [164] mastered *StarCraft II*, reaching Grand Master level and outperforming 99.8% of players. Earlier, AlphaGo [165] defeated the world champion in Go, and similar successes exist in chess. Compared to chess, however, both *StarCraft II* and *Minecraft* are far more complex—real-time strategy involves nearly limitless choices, while sandbox games allow open-ended actions.

5.2.2. Related work

This chapter connects to several themes: competition-based teaching, AGI research, gamification, and machine learning. Competition in education is widely used to foster innovation, teamwork, and motivation [166][167]. Chen, Li, and Wang [168] found that students even preferred competition-based learning over project-based approaches, as in their robot navigation task. While AGI remains a concept with many perspectives on its operation [169][170][171][172], it will likely be trained on tasks such as navigation or object collection, similar to those discussed here.

Gamification is increasingly applied across disciplines, from programming [173][174] to nursing [175] and other fields [176][177], with positive effects when introduced progressively—by analyzing learners and environments, setting objectives, and then applying gamified elements. Yildirim [178] reported that gamification improved student achievement and attitudes.

Recent studies highlight AI-supported gamification. Joshi and Joshi [179] proposed AI-driven, gamified assessments that adapt to learners and applied them in math education using Minecraft. Shamsudin and Teoh [180] showed that Minecraft's AI Copilot aids STEM teaching by enhancing planning, classroom

management, and engagement. Singh and Sun [181] demonstrated Minecraft Education Edition as a metaverse-like platform for STEM, where 40 students reported improved immersion, collaboration, and learning outcomes across Chemistry, Coding, and AI. Their study emphasized gains in learner identity, engagement, belonging, and self-efficacy, confirming Minecraft's potential as an immersive educational tool.

5.2.3. Red Flower Hell

The Red Flower Hell (RFH) is a MALMÖ-based Minecraft challenge (Figure 5.9), created for undergraduate courses in programming and AI to support competitive learning. The core task is for agents to collect as many red flowers as possible before lava descending the hillside reaches them. The XML file defining this gorge-like world is available in the project repository[19].

5.9 Red Flower Hell gorge in Minecraft.

5.2.3.1. Human Intelligence Agents

Before testing AI or heuristic programs, it is useful to see how many flowers a human player can collect in Red Flower Hell. These Human Intelligence (HI) agents are simply players competing in the gorge. The current record is 46 flowers.

[19] https://github.com/nbatfai/RedFlowerHell

5.2.3.2. Software Agents for Introductory Programming Courses

For beginners, agents rely only on simple heuristic rules rather than advanced methods like graph search or Q-learning. Using MALMÖ, students practice programming by building heuristic algorithms from basic instructions, linking coding skills with gameplay. As both the student's knowledge and the agent's strategies improve, performance rises. The best result with a heuristic agent so far is 53 flowers.

5.2.3.2.1. Some Sample Agents

The following simple heuristic example agents have been created for RFH: The GreenPill, Test Subject #40, and MrPoppy. These agents mostly used some variant of the spiral collecting strategy. In the Red Flower Hell phase 3, these agents achieved 15th and 16th places with 28 and 27 flowers collected, in phase 4, they performed better and reached the 3rd place with the Python agent, which collected 9 flowers, the 4th place with the C++ agent, which collected 8 flowers, and the 6th place with a Python agent, which collected 6 flowers.

5.2.3.3. RFH for AI Courses

These agents employ standard AI techniques like Q-learning. In AI courses, where students already know programming basics, they can design more advanced agents than heuristic ones. Since the RFH task and environment are simple, it is well-suited for teaching AI to collect flowers. Students can compare models and parameters, progressing from simple methods (e.g., decision trees) to deep neural networks or graph search as the challenge levels increase. With MALMÖ's Python support, integrating ML models is straightforward, and students can directly observe how their trained models make decisions in the game.

5.2.4. MALMÖ Competitions

Microsoft MALMÖ allows the creation of AI agents whose problem-solving skills can be tested against each other in competitive challenges. While traditional sports have clear winning conditions, AI contests often vary by task. The simplest goal is achieving the highest score, as in MALMÖ challenges like "catch the pig" or our own Red Flower Hell (RFH).

RFH was run in the High-level Programming course during the first semester of 2020, giving students a chance to measure their programming skills through competition. The task—collecting red flowers in a spiral path—is simple in principle, but agents must finish before lava descends the hillside. The competition had four phases, each building on the previous one, and offered a practical example of competition-based learning.

5.2.4.1. First phase

In this phase, students were allowed to "cheat" and use any method to collect flowers. Solutions included querying flower locations and teleporting, blocking lava to gain unlimited time, or, most creatively, replacing lava with water so flowers floated to the agent. As shown in Table 5.16, most students achieved 57–58 flowers compared to the baseline of 13. This round emphasized creativity and testing system boundaries rather than programming skill, and the near-perfect scores illustrate how fully students exploited the open rules.

5.16. Table – Results of the first phase

student_1	58
student_2	58
student_3	58
student_4	58
student_5	58
student_6	58
student_7	58
student_8	58
student_9	58
student_10	58
student_11	58
student_12	58
student_13	58
student_14	57
initial_program	13

5.2.4.2. Second phase

The second phase introduced stricter rules: agents could not extract data, alter the XML map generator, or use absolute movement (imitating key presses), relying only on discrete block-by-block motion. The goal remained collecting red flowers. Most students applied spiral-path strategies, but higher speeds often reduced accuracy, resulting in fewer flowers. As Table 5.17 shows, results ranged from 53 to 17 flowers, proving the restrictions filtered out simplistic

solutions. This phase required genuine algorithmic thinking, forcing students to balance speed and precision.

5.17. Table – Results of the second phase

student_1	53
student_2	41
student_3	39
student_4	37
student_5	28
student_6	26
student_7	24
student_8	23
student_9	22
student_10	21
student_11	20
student_12	19
student_13	17
initial_program	14

5.2.4.3. Third phase

In the third phase, rules remained unchanged, but agents had to qualify by collecting at least 25 flowers, documented with a video recording. The challenge lay in the heavy resource demands of recording, which limited agent performance. As shown in Table 5.18, results ranged from 27 to 50 flowers, with many students submitting multiple attempts. Despite the added technical difficulty, most students met the qualification threshold, which effectively motivated further optimization under hardware constraints.

5.18. Table – Results of the third phase

student_1	50
student_2	48
student_3	44, 37
student_4	43, 34
student_5	40, 26
student_6	38, 29
student_7	35
student_8	33
student_9	32, 29
student_10	31
student_11	31
student_12	30, 28
student_13	30, 29
student_14	30, 26
student_15	30
student_16	30
student_17	30
student_18	30
student_19	30, 28
student_20	30, 29, 28
student_21	29
student_22	29
student_23	29
student_24	29
student_25	28
student_26	28

student_25	27

5.2.4.4. Final phase

The final phase moved agents to a new setting where two slow zombies complicated flower collection. Unlike earlier phases, survival became essential, as failure to defeat the monsters reduced flower counts. Our Python agent placed 3rd and the C++ agent 4th. Table 5.19 shows performance dropped sharply (11–46 flowers), reflecting the added combat challenge. Success required balancing survival with efficient collection—top agents managed both, while weaker ones struggled with hostile threats. This phase demanded more advanced strategies and trade-offs than the previous rounds.

5.19. Table – Results of the final phase

student_1	46
student_2	46
student_3	43
student_4	40
student_5	38
student_6	36
student_7	34
student_8	22
student_9	11

5.2.5. Recommendation for Introductory Programming Courses

As noted earlier, MALMÖ was integrated into the High-level Programming course, allowing students to earn better grades through practice. Since programming is best learned by doing, MALMÖ made it engaging by letting students see their code come alive in Minecraft. Being open-source and supported with tutorials, we recommend MALMÖ to anyone interested in programming, especially beginners.

5.2.5.1. RFH for Undergraduate Courses

In our case, participation in Red Flower Hell was optional, though Python and C++ teaching could easily be built around MALMÖ challenges. The platform offers engaging problems suited to undergraduates, with solutions visualized directly in the game. Many first-year students lack a clear idea of how

code translates into results, and console debugging alone often fails to capture their interest. By contrast, MALMÖ agents provide immediate feedback in a dynamic environment, allowing students to see their code in action, celebrate successes, and stay motivated to practice and improve.

5.2.5.2. RFH for Primary and Secondary School Pupils

RFH and other MALMÖ challenges could also be extended to primary and secondary school students interested in programming. Since MALMÖ supports Python, C++, C#, and Java, it can serve as a platform to teach these languages before university. Benefits include linking coding with gaming to maintain interest, enabling teachers to introduce modern technologies, and allowing students to solve problems with simple heuristic algorithms. Secondary school students often have some familiarity with Python and game engines such as Godot [182]. They may use these tools to create small games, which can encourage them to practice programming in their free time. Although such projects usually do not include machine learning features, they can still be applied in simple games.

5.2.6. Results

This chapter presented a competitive teaching approach using Microsoft's MALMÖ project and Minecraft. Since building agents requires only basic programming skills, the platform is well suited for introductory courses, linking video games with coding to boost student motivation. To test programming and problem-solving abilities, we created the **Red Flower Hell** challenge, structured in four phases of increasing difficulty. Through these stages, students progressed from simple instructions to heuristic algorithms and later to combining multiple skills, preparing them for more advanced topics such as machine learning. Feedback showed that seeing immediate results in a game environment helped them learn faster and stay engaged.

The phased design proved effective in gradually developing skills. Phase 1 encouraged creativity in exploiting open rules, while Phase 2 introduced realistic constraints that required genuine algorithmic thinking. Phase 3 added technical challenges through resource-heavy video recording, and Phase 4 incorporated combat with zombies, demanding strategic balance between survival and collection. As Table results showed, each phase raised complexity and mirrored real-world programming challenges.

Overall, the progression from simple to advanced tasks fostered robust, adaptable solutions and met the pedagogical goals of programming education. The Red Flower Hell challenge demonstrated that competition-based, gamified environments can be both highly educational and motivating for students.

5.2.7. Conclusions

In Red Flower Hell, MALMÖ encouraged problem-solving through trial and error, as testing modified agents was quick and performance could be easily measured by the number of collected flowers. Still, Minecraft's high resource demands slowed development and testing. A major advantage of MALMÖ is its support for popular languages such as Java, C++, Python, and C#.

By Phase 4, simple programming tasks became more complex with the addition of hostile characters alongside lava. This raised a key dilemma: what counts as the "more intelligent" behavior? One strategy was to ignore enemies as long as possible and only fight when necessary, leaving more time for flower collection—this proved more effective. Another was to fight enemies aggressively, but this consumed time and reduced efficiency. The contrast shows how added complexity forces trade-offs between simple but effective strategies and more elaborate, potentially unnecessary behaviors.

5.2.8. Possible improvements

The Red Flower Hell challenge was relatively easy since students knew the fixed map, enemy positions, and flower locations, allowing agents to be optimized for a stable environment. As a result, agents performed well under known conditions but struggled when the setup changed. To improve, randomization could be added—such as varying events, enemy types, or item placement—to encourage agents to adapt. For example, replacing slow zombies with skeleton archers would make running ineffective, or introducing uncertain elements like flowing water could alter movement speed. Multi-agent scenarios could also raise difficulty, requiring players to balance collecting flowers with blocking or defeating opponents.

These extensions would shift RFH from producing narrowly optimized agents to supporting broader behavioral strategies. While the current fixed-map version is suitable for introductory programming, advanced courses could benefit from added complexity and supporting tools like map generators, editors, and automated testers.

Chapter 6

Summary

This dissertation explores the intersection of machine learning (ML), mobile computing, and web-based architectures, with a strong emphasis on performance, scalability, and integration. As ML becomes increasingly embedded in modern applications, the need arises to develop systems that can handle the computational complexity of ML tasks while adapting to resource-constrained environments, such as smartphones and distributed microservice ecosystems. The research addresses critical challenges in training and deploying ML models across heterogeneous platforms, offering practical architectures, implementation strategies, and extensive performance evaluations. The work is based on original publications and real-world experiments that demonstrate the feasibility of applying ML in constrained environments without sacrificing accuracy or responsiveness.

The opening chapter provides a philosophical and technical context for the research. It draws analogies between human cognition and machine-based data processing, underscoring that both rely on data to grow and adapt. The chapter then narrows its focus to machine learning, mobile development, and web applications—three pillars that have individually evolved significantly and whose integration presents new technical frontiers. It also introduces the key motivations behind the thesis: the widespread availability of mobile hardware with underutilized computational power, the benefits of web applications in terms of accessibility and scalability, and the transformative impact of ML on both. The chapter outlines the dissertation's goal: to develop efficient and scalable systems that distribute ML computation across devices and services, optimizing performance for real-world use.

Chapter 2 presents the design and implementation of a distributed ML framework that leverages data parallelism to enable training on mobile devices. A client-server architecture is developed, where the server (a Java-based web service) manages datasets and models, while mobile clients (Android apps) perform local training on dataset partitions. The core components include:

- A Spring-based web service for data slicing, model serialization, and ensemble model construction.
- An Android application that trains neural networks using Deeplearning4J and communicates with the server via HTTP.

- A data exchange protocol based on serialized byte arrays to ensure compatibility and integrity during model transfer.

Experimental evaluation demonstrates that, although mobile devices have limited resources, they can successfully contribute to model training via data parallelism. When submodels trained on slices are aggregated into an ensemble, the resulting model achieves accuracy comparable to that of a model trained on the entire dataset using a traditional desktop system. This chapter highlights the feasibility and value of utilizing idle mobile computation power for ML tasks.

Chapter 3 investigates how existing Java-based ML libraries can be adapted for use in the Android ecosystem. Given the limitations of Python on Android, the study focuses on Weka, SMILE, and Oracle Tribuo—three prominent Java libraries. The chapter documents the process of porting these libraries to Android, overcoming issues related to unsupported Java features and Android-specific constraints. An experimental benchmarking application is developed to compare runtime performance, memory usage, CPU load, and battery consumption across the libraries. Key findings include:

- **Weka** offers the best overall efficiency in terms of memory and energy consumption, making it suitable for real-world mobile ML applications.
- **SMILE** is the fastest in some specific use cases (e.g., K-means clustering) but is the most CPU-intensive.
- **Tribuo** provides modern APIs and extensibility, but exhibits inconsistent performance on larger datasets.

The chapter concludes that while all three libraries can function on Android with some limitations, Weka is best suited for general-purpose mobile ML tasks. These findings provide guidance for developers who aim to integrate ML directly into Android apps without relying on external services.

Chapter 4 addresses the challenge of integrating heterogeneous ML models into scalable web systems. It proposes a microservice-based architecture that enables ensemble learning across models trained in different environments and programming languages. The architecture supports:

- Wrapping ML libraries (e.g., Scikit-learn, Weka, TensorFlow) in independent web services.
- Two architectural variants: a **direct** model where services are queried individually, and a **gateway** model where requests are routed through an aggregator.

- Dynamic model ensembling, allowing for majority voting or weighted voting strategies.

This framework enables seamless interoperability between components written in different languages and facilitates the reuse of models across applications. The study emphasizes the flexibility and maintainability of microservice architectures, particularly in complex ML workflows. Challenges such as communication overhead, network latency, and increased architectural complexity are acknowledged and addressed with design recommendations.

Chapter 5 demonstrates applications related to the field of machine learning in two innovative scenarios:

The first section presents a unique study comparing how humans and neural networks perceive numerosity—i.e., the number of items in a set. The experiments are inspired by cognitive psychology and investigate how both biological and artificial systems can recognize small quantities without explicit counting. Several variations of the SMNIST dataset are introduced and tested across deep learning models. The results show that while machines can learn to approximate subitizing, they rely heavily on training data and lack the innate efficiency of the human Object File System (OFS). This comparison adds a cognitive-scientific dimension to the thesis and deepens our understanding of ML model limitations.

The second chapter introduces a gamified educational framework for teaching programming, implemented in Minecraft using the MALMÖ API. Students program agents to navigate complex tasks (e.g., collecting red flowers while avoiding traps), learning about decision trees, heuristics, and reinforcement learning along the way. The approach increases student engagement and performance in introductory programming courses, demonstrating the power of interactive, AI-driven educational tools. It also exemplifies how ML concepts can be integrated into game-based learning environments.

Acknowledgements

I would like to thank many people for their support, as I believe that the impressions left by personal and professional relationships have shaped my career in this direction, ultimately making the completion of this dissertation possible.

Special thanks go to Márton Ispány, who was not only my supervisor but also my colleague. His comprehensive professional knowledge and experience gave me confidence while writing my papers, as I knew I could always turn to him for advice. My choice of topic and interest in the field are also thanks to the data mining courses he taught. After the first few lectures, I knew this was the area I wanted to work in for the coming years.
I would like to thank my wife, Eszter Szabó-Hirku, for her steadfast support throughout this long journey. I am equally grateful to my children, Zoé and Nándi, who, through fatherhood, revealed entirely new perspectives on life, giving me renewed energy to write this dissertation. I owe a great deal to my brother, Bence, whose determination always encouraged me to persevere, and whose professional curiosity sparked my own interest in new technologies. As both a sibling and a colleague, he significantly contributed to the success of my studies and research.

I thank my parents, Ildikó and Zsolt, for nurturing my interest in IT since childhood and for doing everything they could to help me achieve my goals. I also thank my grandparents, through whom I came to understand the value of hard work and perseverance, which has allowed me to reach any goal I set. My gratitude extends to the rest of my family as well, whose valuable advice and diverse life paths helped guide me toward making the right decisions.

I am thankful to Jókai Mór Reformed Primary School and the Széchenyi István Secondary School of Economics and IT, as it was in these institutions that the spark of love for computer science was ignited. I would like to express my gratitude to József Bezzeg and Tamás Komoróczy for the outstanding, student-centered IT education I received in the years before university.

Thanks also go to the University of Debrecen, the Faculty of Informatics, and the Doctoral School of Informatics for allowing me to study, work, and conduct research. A significant part of my life and career is tied to these institutions.

To sustain a lifelong love of IT, I needed teachers and lecturers who were excellent professionals, passionate about their field, and committed to helping students become great professionals themselves. At the Faculty of Informatics,

many lecturers helped shape my professional outlook in their own unique ways. I would like to thank Péter Jeszenszky, Piroska Biró, Tibor Balla, Attila Adamkó, János Pánovics, Anikó Vágner, György Terdik, László Szathmáry, Gábor Halász, and István Fazekas.

I am especially grateful to Norbert Bátfai, who, during my studies, introduced me to real-world challenges and practical problems in the field. Later, he was one of those who introduced me to academic research, for which I will be forever thankful. His unconventional projects—such as Binfa, Robot Soccer, Robocar Championship, Samu, and RFH—left a lasting impression on every student.

Throughout my professional life, I have met several outstanding experts who introduced me to the more practical side of IT and programming, highlighting the challenges they contain. I am grateful to Zsolt Kontra and Róbert Temesi, who introduced me to managing databases and systems of a scale I could not have previously imagined. I also thank Attila Kovács and Zoltán Varga from Baromfiudvar 2002 Kft., who broadened my perspective beyond what I had learned in school and showcased the diversity of IT. From the experts working there, I learned about numerous technologies, tools, and methodologies, all of which made my teaching more effective. The machine learning experiments I conducted there had a significant impact on my scientific work as well.

Bibliography

[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] F. Brakel, U. Odyurt, and A.-L. Varbanescu, “Model Parallelism on Distributed Infrastructure: A Literature Review from Theory to LLM Case-Studies,” Mar. 06, 2024, arXiv: arXiv:2403.03699. doi: 10.48550/arXiv.2403.03699.

[28] C. Peng, “Comparison of Distributed and Parallel Machine Learning: Efficiency and Effectiveness in Large-Scale Data Processing,” ACE, vol. 96, no. 1, pp. 175–180, Nov. 2024, doi: 10.54254/2755-2721/96/20241279.

[29] B. Chatterjee, “Distributed Machine Learning,” Proceedings of the 25th International Conference on Distributed Computing and Networking. ACM, pp. 4–7, Jan. 04, 2024. doi: 10.1145/3631461.3632516.

[30] R. Johnson et al., Spring Framework – Reference Documentation, Spring Framework 4.3.2.RELEASE. [Online]. Available: https://docs.spring.io/spring-framework/docs/4.3.2.RELEASE/spring-framework-reference/html/

[31] H. Li, A. Kadav, E. Kruus, and C. Ungureanu, "MALT," Proceedings of the Tenth European Conference on Computer Systems. ACM, pp. 1–16, Apr. 17, 2015. doi: 10.1145/2741948.2741965.

[32] E. P. Xing et al., "Petuum: A New Platform for Distributed Machine Learning on Big Data," IEEE Trans. Big Data, vol. 1, no. 2, pp. 49–67, June 2015, doi: 10.1109/tbdata.2015.2472014.

[33] "UCI Machine Learning Repository." https://archive.ics.uci.edu/ml/datasets/record+linkage+comparison+patterns, (Last visited 24 August 2025).

[34] M. Sariyar, A. Borg, and K. Pommerening, "Controlling false match rates in record linkage using extreme value theory," Journal of Biomedical Informatics, vol. 44, no. 4, pp. 648–654, Feb. 2011, doi: 10.1016/j.jbi.2011.02.008.

[35] 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.

[36] 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.

[37] 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

[38] 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.

[39] 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.

[40] 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

[41] 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.

[42] S. Zhu, T. Voigt, F. Rahimian, and J. Ko, “On-device Training: A First Overview on Existing Systems,” ACM Transactions on Sensor Networks., vol. 20, no. 6, pp. 1–39, Oct. 2024, doi: 10.1145/3696003.

[43] J. Moon et al., “A New Frontier of AI: On-Device AI Training and Personalization,” Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice. ACM, pp. 323–333, Apr. 14, 2024. doi: 10.1145/3639477.3639716.

[44] 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.

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

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

[47] 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.

[48] 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.

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

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

[51] 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.

[52] R. Marsan, “Weka-for-Android”, 2011 GitHub https://github.com/rjmarsan/Weka-for-Android.

[53] R. Fischer, “Iris Dataset”, 1988, https://archive.ics.uci.edu/ml/datasets/iris.

[54] “Google trends”, 2025 https://trends.google.com/trends/

[55] “The state of developer ecosystem in 2024 infographic.” 2025 https://www.jetbrains.com/lp/devecosystem-2024/

[56] T. G. Dietterich, “Ensemble Methods in Machine Learning,” Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 1–15, 2000. doi: 10.1007/3-540-45014-9_1.

[57] C. Zhang, Y. Xie, H. Bai, B. Yu, W. Li, and Y. Gao, “A survey on federated learning,” Knowledge-Based Systems, vol. 216, p. 106775, Mar. 2021, doi: 10.1016/j.knosys.2021.106775.

[58] 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.

[59] 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.

[60] 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.

[61] 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.

[62] 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.

[63] 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.

[64] 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.

[65] 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.

[66] 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.

[67] 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.

[68] 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.

[69] 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.

[70] S. Moreschini et al., "AI Techniques in the Microservices Life-Cycle: a Systematic Mapping Study," Computing, vol. 107, no. 4, Mar. 2025, doi: 10.1007/s00607-025-01432-z.

[71] P. De Rosa, Y.-D. Bromberg, P. Felber, D. Mvondo, and V. Schiavoni, "On the Cost of Model-Serving Frameworks: An Experimental Evaluation,"

2024 IEEE International Conference on Cloud Engineering (IC2E). IEEE, pp. 221–232, Sept. 24, 2024. doi: 10.1109/ic2e61754.2024.00032.

[72] 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.

[73] 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.

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

[75] 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.

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

[77] Qian Zhang, Hanyue Chu, Mingyu Li, and Xiaohui Hu, “A unified API gateway for high availability clusters,” Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC). IEEE, pp. 2321–2325, Dec. 2013. doi: 10.1109/mec.2013.6885428.

[78] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proc. IEEE, vol. 86, no. 11, pp. 2278–2324, 1998, doi: 10.1109/5.726791.

[79] Y. LeCun, C. Cortes, and C. J. C. Burges, “The MNIST database of handwritten digits,” 1998 http://yann.lecun.com/exdb/mnist/.

[80] R. E. Núñez, “Is There Really an Evolved Capacity for Number?,” Trends in Cognitive Sciences, vol. 21, no. 6, pp. 409–424, June 2017, doi: 10.1016/j.tics.2017.03.005.

[81] M. Dacke and M. V. Srinivasan, “Evidence for counting in insects,” Animal Cognition, vol. 11, no. 4, pp. 683–689, May 2008, doi: 10.1007/s10071-008-0159-y.

[82] H. M. Ditz and A. Nieder, “Numerosity representations in crows obey the Weber–Fechner law,” Proceedings of the Royal Society B, vol. 283, no. 1827, p. 20160083, Mar. 2016, doi: 10.1098/rspb.2016.0083.

[83] L. Feigenson, S. Carey, and M. Hauser, “The Representations Underlying Infants’ Choice of More: Object Files Versus Analog Magnitudes,” Psychological Science, vol. 13, no. 2, pp. 150–156, Mar. 2002, doi: 10.1111/1467-9280.00427.

[84] L. Feigenson, S. Dehaene, and E. Spelke, “Core systems of number,” Trends in Cognitive Sciences, vol. 8, no. 7, pp. 307–314, July 2004, doi: 10.1016/j.tics.2004.05.002.

[85] D. C. Geary, “Reflections of evolution and culture in children’s cognition: Implications for mathematical development and instruction,” American Psychologist, vol. 50, no. 1, pp. 24–37, Jan. 1995, doi: 10.1037/0003-066x.50.1.24.

[86] D. C. Geary, “From infancy to adulthood: the development of numerical abilities,” European Child and Adolescent Psychiatry, vol. 9, no. S2, pp. S11–S16, June 2000, doi: 10.1007/s007870070004.

[87] D. C. Hyde, “Two Systems of Non-Symbolic Numerical Cognition,” Frontiers in Human Neuroscience, vol. 5, 2011, doi: 10.3389/fnhum.2011.00150.

[88] A. Nieder, “The neuronal code for number,” Nature Reviews Neuroscience, vol. 17, no. 6, pp. 366–382, May 2016, doi: 10.1038/nrn.2016.40.

[89] H. Davis and R. Pérusse, “Numerical competence in animals: Definitional issues, current evidence, and a new research agenda,” Behavioral and Brain Sciences, vol. 11, no. 4, pp. 561–579, Dec. 1988, doi: 10.1017/s0140525x00053437.

[90] R. F. Thompson, K. S. Mayers, R. T. Robertson, and C. J. Patterson, “Number Coding in Association Cortex of the Cat,” Science, vol. 168, no. 3928, pp. 271–273, Apr. 1970, doi: 10.1126/science.168.3928.271.

[91] S. T. Boysen and G. G. Berntson, “Numerical competence in a chimpanzee (Pan troglodytes),” Journal of Comparative Psychology, vol. 103, no. 1, pp. 23–31, 1989, doi: 10.1037/0735-7036.103.1.23.

[92] H. Davis and J. Memmott, “Counting behavior in animals: A critical evaluation,” Psychological Bulletin, vol. 92, no. 3, pp. 547–571, Nov. 1982, doi: 10.1037/0033-2909.92.3.547.

[93] I. M. Pepperberg, “Evidence for Conceptual Quantitative Abilities in the African Grey Parrot: Labeling of Cardinal Sets,” Ethology, vol. 75, no. 1, pp. 37–61, Jan. 1987, doi: 10.1111/j.1439-0310.1987.tb00641.x.

[94] S. E. Antell and D. P. Keating, “Perception of Numerical Invariance in Neonates,” Child Development, vol. 54, no. 3, p. 695, June 1983, doi: 10.2307/1130057.

[95] P. Starkey, “The early development of numerical reasoning,” Cognition, vol. 43, no. 2, pp. 93–126, May 1992, doi: 10.1016/0010-0277(92)90034-f.

[96] P. Starkey, E. S. Spelke, and R. Gelman, “Detection of Intermodal Numerical Correspondences by Human Infants,” Science, vol. 222, no. 4620, pp. 179–181, Oct. 1983, doi: 10.1126/science.6623069.

[97] P. Starkey, E. S. Spelke, and R. Gelman, “Numerical abstraction by human infants,” Cognition, vol. 36, no. 2, pp. 97–127, Aug. 1990, doi: 10.1016/0010-0277(90)90001-z.

[98] L. M. Trick, “Chapter 7 A Theory Of Enumeration That Grows Out Of A General Theory Of Vision: Subitizing, Counting, And Finsts,” Advances in Psychology. Elsevier, pp. 257–299, 1992. doi: 10.1016/s0166-4115(08)60889-4.

[99] E. Van Loosbroek and A. W. Smitsman, “Visual perception of numerosity in infancy,” Developmental Psychology, vol. 26, no. 6, pp. 916–922, Nov. 1990, doi: 10.1037/0012-1649.26.6.911.b.

[100] M. Zorzi and A. Testolin, “An emergentist perspective on the origin of number sense,” Philosophical Transactions of the Royal Society B, vol. 373, no. 1740, p. 20170043, Jan. 2018, doi: 10.1098/rstb.2017.0043.

[101] J. Halberda, M. M. M. Mazzocco, and L. Feigenson, “Individual differences in non-verbal number acuity correlate with maths achievement,” Nature, vol. 455, no. 7213, pp. 665–668, Sept. 2008, doi: 10.1038/nature07246.

[102] D. Odic, M. E. Libertus, L. Feigenson, and J. Halberda, “Developmental change in the acuity of approximate number and area representations,” Developmental Psychology, vol. 49, no. 6, pp. 1103–1112, 2013, doi: 10.1037/a0029472.

[103] M. Piazza, “Neurocognitive start-up tools for symbolic number representations,” Trends in Cognitive Sciences, vol. 14, no. 12, pp. 542–551, Dec. 2010, doi: 10.1016/j.tics.2010.09.008.

[104] M. Piazza et al., “Developmental trajectory of number acuity reveals a severe impairment in developmental dyscalculia,” Cognition, vol. 116, no. 1, pp. 33–41, July 2010, doi: 10.1016/j.cognition.2010.03.012.

[105] M. Piazza, V. Izard, P. Pinel, D. Le Bihan, and S. Dehaene, “Tuning Curves for Approximate Numerosity in the Human Intraparietal Sulcus,”

Neuron, vol. 44, no. 3, pp. 547–555, Oct. 2004, doi: 10.1016/j.neuron.2004.10.014.

[106] M. Piazza, P. Pinel, D. Le Bihan, and S. Dehaene, “A Magnitude Code Common to Numerosities and Number Symbols in Human Intraparietal Cortex,” Neuron, vol. 53, no. 2, pp. 293–305, Jan. 2007, doi: 10.1016/j.neuron.2006.11.022.

[107] S. Dehaene and J.-P. Changeux, “Development of Elementary Numerical Abilities: A Neuronal Model,” Journal of Cognitive Neuroscience, vol. 5, no. 4, pp. 390–407, 1993, doi: 10.1162/jocn.1993.5.4.390.

[108] E. M. Duncan and C. E. McFarland, “Isolating the effects of symbolic distance, and semantic congruity in comparative judgments: An additive-factors analysis,” Memory and Cognition, vol. 8, no. 6, pp. 612–622, Nov. 1980, doi: 10.3758/bf03213781.

[109] G. T. Fechner, H. E. Adler, D. H. Howes, and E. G. Boring, Elements of psychophysics. 1966. [Online]. Available: http://ci.nii.ac.jp/ncid/BA05559164

[110] M. P. van Oeffelen and P. G. Vos, “A probabilistic model for the discrimination of visual number,” Perception and Psychophysics, vol. 32, no. 2, pp. 163–170, Mar. 1982, doi: 10.3758/bf03204275.

[111] S. Dehaene, G. Dehaene-Lambertz, and L. Cohen, “Abstract representations of numbers in the animal and human brain,” Trends in Neurosciences, vol. 21, no. 8, pp. 355–361, Aug. 1998, doi: 10.1016/s0166-2236(98)01263-6.

[112] T. Verguts and W. Fias, “Representation of Number in Animals and Humans: A Neural Model,” Journal of Cognitive Neuroscience, vol. 16, no. 9, pp. 1493–1504, Nov. 2004, doi: 10.1162/0898929042568497.

[113] E. Szathmáry, “The evolution of replicators,” Philosophical transactions of the Royal Society of London Series B: Biological Sciences, vol. 355, no. 1403, pp. 1669–1676, Nov. 2000, doi: 10.1098/rstb.2000.0730.

[114] A. Szilágyi, I. Zachar, A. Fedor, H. P. de Vladar, and E. Szathmáry, “Breeding novel solutions in the brain: A model of Darwinian neurodynamics,” F1000 Research, vol. 5, p. 2416, June 2017, doi: 10.12688/f1000research.9630.2.

[115] J. Von Neumann, The computer and the brain. 1958. [Online]. Available: http://ci.nii.ac.jp/ncid/BA18777521

[116] D. E. Knuth, “The art of computer programming. Vol.2: Seminumerical algorithms,” Literacy, Jan. 1981, [Online]. Available: http://ui.adsabs.harvard.edu/abs/1981acp..book.....K/abstract

[117] J. von Neumann, “The general and logical theory of automata,” in John von Neumann: Collected Works. Vol. V: Design of Computers, Theory of Automata and Numerical Analysis, A. H. Taub, Ed. Oxford, U.K.: Pergamon Press, 1963.

[118] D. Castelvecchi, “Can we open the black box of AI?,” Nature, vol. 538, no. 7623, pp. 20–23, Oct. 2016, doi: 10.1038/538020a.

[119] Y. LeCun et al., “Backpropagation Applied to Handwritten Zip Code Recognition,” Neural Computation, vol. 1, no. 4, pp. 541–551, Dec. 1989, doi: 10.1162/neco.1989.1.4.541.

[120] M. D. Zeiler and R. Fergus, “Visualizing and Understanding Convolutional Networks,” Nov. 28, 2013, arXiv: arXiv:1311.2901. doi: 10.48550/arXiv.1311.2901.

[121] A. Krizhevsky, “Learning Multiple Layers of Features from Tiny Images,” 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[122] V. Mnih et al., “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[123] D. Silver et al., “Mastering the game of Go without human knowledge,” Nature, vol. 550, no. 7676, pp. 354–359, Oct. 2017, doi: 10.1038/nature24270.

[124] M. Jaderberg et al., “Human-level performance in first-person multiplayer games with population-based deep reinforcement learning,” Science, vol. 364, no. 6443, pp. 859–865, May 2019, doi: 10.1126/science.aau6249.

[125] O. Vinyals et al., “StarCraft II: A New Challenge for Reinforcement Learning,” Aug. 16, 2017, arXiv: arXiv:1708.04782. doi: 10.48550/arXiv.1708.04782.

[126] G. Kahn, “The semantics of a simple language for parallel programming,” IFIP Congress, pp. 471–475, Jan. 1974, [Online]. Available: http://perso.ensta-paristech.fr/~chapoutot/various/kahn_networks.pdf

[127] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” Bulletin of Mathematical Biophysics, vol. 5, no. 4, pp. 115–133, Dec. 1943, doi: 10.1007/bf02478259.

[128] F. Rosenblatt, “The perceptron: A probabilistic model for information storage and organization in the brain,” Psychological Review, vol. 65, no. 6, pp. 386–408, 1958, doi: 10.1037/h0042519.

[129] M. Johnson, K. Hofmann, T. Hutton, and D. Bignell, “The Malmo platform for artificial intelligence experimentation,” International Joint Conference on Artificial Intelligence, pp. 4246–4247, Jul. 2016, [Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/07/johnson-malmo-platform-camera-ready.pdf

[130] C. Beattie et al., “DeepMind Lab,” Dec. 13, 2016, arXiv: arXiv:1612.03801. doi: 10.48550/arXiv.1612.03801.

[131] J. Hernández-Orallo et al., “A New AI Evaluation Cosmos: Ready to Play the Game?,” AI Magazine, vol. 38, no. 3, pp. 66–69, Sept. 2017, doi: 10.1609/aimag.v38i3.2748.

[132] K. Arulkumaran, A. Cully, and J. Togelius, “AlphaStar,” Proceedings of the Genetic and Evolutionary Computation Conference Companion. ACM, pp. 314–315, July 13, 2019. doi: 10.1145/3319619.3321894.

[133] M. Jaderberg et al., “Population Based Training of Neural Networks,” 2017, arXiv. doi: 10.48550/ARXIV.1711.09846.

[134] T. Weng, J. Wang, W. Jiang, and Z. Ming, “VisNumBench: Evaluating Number Sense of Multimodal Large Language Models,” July 31, 2025, arXiv: arXiv:2503.14939. doi: 10.48550/arXiv.2503.14939.

[135] A. Testolin, K. Hou, and M. Zorzi, “Visual Enumeration Remains Challenging for Multimodal Generative AI,” July 28, 2025, arXiv: arXiv:2402.03328. doi: 10.48550/arXiv.2402.03328.

[136] V. Izard and S. Dehaene, “Calibrating the mental number line,” Cognition, vol. 106, no. 3, pp. 1221–1247, Mar. 2008, doi: 10.1016/j.cognition.2007.06.004.

[137] S. K. Revkin, M. Piazza, V. Izard, L. Cohen, and S. Dehaene, “Does Subitizing Reflect Numerical Estimation?,” Psychol Sci, vol. 19, no. 6, pp. 607–614, June 2008, doi: 10.1111/j.1467-9280.2008.02130.x.

[138] X. Wu, X. Zhang, and X. Shu, “Cognitive Deficit of Deep Learning in Numerosity,” Nov. 11, 2018, arXiv: arXiv:1802.05160. doi: 10.48550/arXiv.1802.05160.

[139] I. Stoianov and M. Zorzi, “Emergence of a ‘visual number sense’ in hierarchical generative models,” Nature Neuroscience, vol. 15, no. 2, pp. 194–196, Jan. 2012, doi: 10.1038/nn.2996.

[140] J. D. Roitman, E. M. Brannon, and M. L. Platt, “Monotonic Coding of Numerosity in Macaque Lateral Intraparietal Area,” PLoS Biology, vol. 5, no. 8, p. e208, July 2007, doi: 10.1371/journal.pbio.0050208.

[141] S. Chen, Z. Zhou, M. Fang, and J. McClelland, “Can generic neural networks estimate numerosity like humans,” Cognitive Science, Jan. 2018, [Online]. Available: http://dblp.uni-trier.de/db/conf/cogsci/cogsci2018.html#ChenZFM18

[142] A. Testolin, S. Dolfi, M. Rochus, and M. Zorzi, “Perception of visual numerosity in humans and machines,” July 16, 2019, arXiv: arXiv:1907.06996. doi: 10.48550/arXiv.1907.06996.

[143] T. Hannagan, A. Nieder, P. Viswanathan, and S. Dehaene, “A random-matrix theory of the number sense,” Philosophical Transactions of the Royal Society B, vol. 373, no. 1740, p. 20170253, Jan. 2018, doi: 10.1098/rstb.2017.0253.

[144] A. Nieder, “Representing Something Out of Nothing: The Dawning of Zero,” Trends in Cognitive Sciences, vol. 20, no. 11, pp. 830–842, Nov. 2016, doi: 10.1016/j.tics.2016.08.008.

[145] M. Donald, “Origins of the modern mind: three stages in the evolution of culture and cognition,” Choice Reviews Online, vol. 29, no. 07, pp. 29–4184, Mar. 1992, doi:10.5860/choice.29-4184.

[146] “Norbert Bátfai / SMNIST GitLab,” GitLab. https://gitlab.com/nbatfai/smnist

[147] J. Lonnemann, J. Linkersdörfer, M. Hasselhorn, and S. Lindberg, “Developmental changes in the association between approximate number representations and addition skills in elementary school children,” Frontiers in Psychology, vol. 4, 2013, doi: 10.3389/fpsyg.2013.00783.

[148] L. He, K. Zhou, T. Zhou, S. He, and L. Chen, “Topology-defined units in numerosity perception,” Proceedings of the National Academy of Sciences U.S.A., vol. 112, no. 41, Sept. 2015, doi: 10.1073/pnas.1512408112.

[149] T. Gebuis and B. Reynvoet, “The interplay between nonsymbolic number and its continuous visual properties,” Journal of Experimental Psychology: General, vol. 141, no. 4, pp. 642–648, 2012, doi: 10.1037/a0026218.

[150] Y. Tang, Tensorflow, “GitHub - tensorflow/tensorflow: An Open Source Machine Learning Framework for Everyone,” GitHub. https://github.com/tensorflow/tensorflow

[151] F. Chollet et al., Keras-Team, “GitHub - keras-team/keras: Deep Learning for humans,” GitHub. https://github.com/keras-team/keras

[152] A. Paszke et al., “Automatic differentiation in PyTorch,” NIPS Workshop Autodiff Submission, Oct. 2017, [Online]. Available: http://openreview.net/pdf?id=BJJsrmfCZ

[153] deeplearning4j, “GitHub - deeplearning4j/deeplearning4j-examples: Deeplearning4j Examples (DL4J, DL4J Spark, DataVec),” GitHub. https://github.com/deeplearning4j/dl4j-examples

[154] Tensorflow, “GitHub - tensorflow/swift-models: Models and examples built with Swift for TensorFlow,” GitHub. https://github.com/tensorflow/swift-models

[155] N. Y. Du, W. Wang, and L. Wang, “Hierarchical recurrent neural network for skeleton based action recognition,” 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, pp. 1110–1118, June 2015. doi: 10.1109/cvpr.2015.7298714.

[156] J. Li, M.-T. Luong, and D. Jurafsky, “A Hierarchical Neural Autoencoder for Paragraphs and Documents,” June 06, 2015, arXiv: arXiv:1506.01057. doi: 10.48550/arXiv.1506.01057.

[157] “Apache MXNet,” Apache MXNet. https://mxnet.apache.org/

[158] S. Dieleman et al., Lasagne: First release. Zenodo, 2015. doi: 10.5281/ZENODO.27878.

[159] V. Mnih et al., “Playing Atari with Deep Reinforcement Learning,” Dec. 19, 2013, arXiv: arXiv:1312.5602. doi: 10.48550/arXiv.1312.5602.

[160] W. H. Guss et al., “The MineRL 2019 Competition on Sample Efficient Reinforcement Learning using Human Priors,” Jan. 19, 2021, arXiv: arXiv:1904.10079. doi: 10.48550/arXiv.1904.10079.

[161] J. Hsu, “AI takes on popular Minecraft game in machine-learning contest,” Nature, vol. 575, no. 7784, pp. 583–584, Nov. 2019, doi: 10.1038/d41586-019-03630-0.

[162] D. Perez-Liebana et al., “The Multi-Agent Reinforcement Learning in MalmÖ (MARLÖ) Competition,” Apr. 11, 2025, arXiv: arXiv:1901.08129. doi: 10.48550/arXiv.1901.08129.

[163] N. Bátfai, C. Csukonyi, D. Papp, C. Hermann, E. Deákné Osvald, and K. Győri, “A DEAC-Hackersesport szakosztály mesterséges intelligencia oktatási és kutatási elképzelése a Minecraftban,” Mesterséges intelligencia, vol. 2, no. 1, pp. 95–109, 2020, doi: 10.35406/mi.2020.1.95.

[164] O. Vinyals et al., “Grandmaster level in StarCraft II using multi-agent reinforcement learning,” Nature, vol. 575, no. 7782, pp. 350–354, Oct. 2019, doi: 10.1038/s41586-019-1724-z.

[165] D. Silver et al., “Mastering the game of Go without human knowledge,” Nature, vol. 550, no. 7676, pp. 354–359, Oct. 2017, doi: 10.1038/nature24270.

[166] H. F. Ladd and E. B. Fiske, “Does Competition Improve Teaching and Learning? Evidence from New Zealand,” Educational Evaluation and Policy Analysis, vol. 25, no. 1, pp. 97–112, Mar. 2003, doi: 10.3102/01623737025001095.

[167] N. L. Huang, N. L. Dai, N. M. Huang, N. H. Wang, and N. B. Guo, “Promoting teaching by competition and combining teaching with competition,” International Conference on Computer Science and Service System (CSSS), 2011, pp. 3383–3385. doi: 10.1109/csss.2011.5972062

[168] D. Chen, Z. Li, and T. Wang, “Exploration and practice: A competition based project practice teaching mode,” Mechatronics, vol. 24, no. 2, pp. 128–138, Mar. 2014, doi: 10.1016/j.mechatronics.2013.12.009.

[169] S. S. Adams et al., “Mapping the Landscape of Human-Level Artificial General Intelligence,” AI Magazine, vol. 33, no. 1, pp. 25–41, Mar. 2012, doi: 10.1609/aimag.v33i1.2322.

[170] B. Goertzel, “Artificial General Intelligence: Concept, State of the Art, and Future Prospects,” Journal of Artificial General Intelligence, vol. 5, no. 1, pp. 1–48, Dec. 2014, doi: 10.2478/jagi-2014-0001.

[171] B. Goertzel and C. Pennachin, Eds., Artificial General Intelligence. Springer Berlin Heidelberg, 2007. doi: 10.1007/978-3-540-68677-4.

[172] B. Goertzel, Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms. in Frontiers in Artificial Intelligence and Applications Series, no. v. 157. Amsterdam: IOS Press, 2007.

[173] M. Carreno-Leon, A. Sandoval-Bringas, F. Alvarez-Rodriguez, and Y. Camacho-Gonzalez, “Gamification technique for teaching programming,” 2018 IEEE Global Engineering Education Conference (EDUCON). IEEE, pp. 2009–2014, Apr. 2018. doi: 10.1109/educon.2018.8363482.

[174] C. S. G. Gonzalez and A. M. Carreno, “Methodological proposal for gamification in the computer engineering teaching,” 2014 International Symposium on Computers in Education (SIIE). IEEE, pp. 29–34, Nov. 2014. doi: 10.1109/siie.2014.7017700.

[175] C. Day-Black, E. B. Merrill, L. Konzelman, T. T. Williams, and N. Hart, “Gamification: an innovative Teaching-Learning strategy for the digital nursing students in a community health nursing course,” PubMed, vol. 26, no. 4, pp. 90–4, Jan. 2015, [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/26665503

[176] I. R. Management Association, Ed., “Gamification in Education” IGI Global, 2018. doi: 10.4018/978-1-5225-5198-0.

[177] S. Villagrasa, D. Fonseca, E. Redondo, and J. Duran, “Teaching Case of Gamification and Visual Technologies for Education,” Journal of Cases on Information Technology, vol. 16, no. 4, pp. 38–57, Oct. 2014, doi: 10.4018/jcit.2014100104.

[178] I. Yildirim, “The effects of gamification-based teaching practices on student achievement and students’ attitudes toward lessons,” The Internet and Higher Education, vol. 33, pp. 86–92, Apr. 2017, doi: 10.1016/j.iheduc.2017.02.002.

[179] Namya Joshi and M. Joshi, “Gamified AI-Driven Assessments,” Unpublished, 2024, doi: 10.13140/RG.2.2.11415.69289.

[180] N. M. Shamsudin and T. S. Hoon, “The Use of AI Copilot in Minecraft Education Edition for Teacher Training: Enhancing Pedagogical Practices and STEM Learning,” Advances in Computer Science Research. Atlantis Press International BV, pp. 525–531, 2024. doi: 10.2991/978-94-6463-589-8_49.

[181] M. Singh and D. Sun, “Evaluating Minecraft as a game-based metaverse platform: exploring gaming experience, social presence, and STEM outcomes,” Interactive Learning Environments, pp. 1–23, Feb. 2025, doi: 10.1080/10494820.2025.2459200.

[182] J. Linietsky, A. Manzur, R. Verschelde, et al., “Godot Engine - Free and open source 2D and 3D game engine,” Godot Engine. https://godotengine.org/

Own publications

[O1] 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.

[O2] M. Szabó, ”Machine Learning on Android with Oracle Tribuo, SMILE and Weka,” CEUR Workshop Proceedings, vol. 2874, pp. 176-186, 2021.

[O3] 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.

[O4] 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.

[O5] 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.

Összefoglaló

Ez a disszertáció a gépi tanulás (ML), a mobil számítástechnika és a webalapú architektúrák metszéspontját vizsgálja, különös tekintettel a teljesítményre, a skálázhatóságra és az integrációra. Ahogy a gépi tanulás egyre inkább beépül a modern alkalmazásokba, nő az igény olyan rendszerek fejlesztésére, amelyek képesek kezelni az ML-feladatok számítási komplexitását, miközben alkalmazkodnak az erőforrás-korlátozott környezetekhez, mint például az okostelefonok és az elosztott mikroszolgáltatás-alapú rendszerek. A kutatás kritikus kihívásokkal foglalkozik az ML-modellek heterogén platformokon történő betanítása és telepítése kapcsán, gyakorlati architektúrákat, megvalósítási stratégiákat és kiterjedt teljesítményelemzéseket kínálva. A dolgozat eredeti publikációkon és valós kísérleteken alapul, amelyek igazolják, hogy az ML korlátozott környezetekben is hatékonyan alkalmazható a pontosság vagy a reakcióidő feláldozása nélkül.

A bevezető fejezet filozófiai és technikai kontextust biztosít a kutatáshoz. Párhuzamot von az emberi megismerés és a gépi adatfeldolgozás között, hangsúlyozva, hogy mindkettő fejlődéséhez az adatok szolgálnak alapul. A fejezet ezt követően leszűkíti a fókuszt a gépi tanulásra, a mobil fejlesztésre és a webes alkalmazásokra – három olyan pillérre, amelyek önmagukban is jelentős fejlődésen mentek keresztül, integrációjuk pedig új technikai kihívásokat és lehetőségeket teremt. Bemutatja a dolgozat fő motivációját is: a mobil hardverek széles körű elérhetőségét és kihasználatlanságát, a webes alkalmazások előnyeit (hozzáférhetőség, skálázhatóság), valamint az ML átalakító hatását. A fejezet felvázolja a dolgozat célját is: olyan hatékony és skálázható rendszerek fejlesztését, amelyek az ML számítási feladatait eszközök és szolgáltatások között képesek elosztani a teljesítmény optimalizálása érdekében.

A 2. fejezet egy olyan elosztott ML-keretrendszer megtervezését és megvalósítását mutatja be, amely adatpárhuzamosságot használ ki a mobil eszközökön történő tanításhoz. A szerver-kliens architektúra keretében a szerver (egy Java-alapú webszolgáltatás) kezeli az adatokat és a modelleket, míg a mobil kliensek (Android alkalmazások) helyi tanítást végeznek az adathalmazok szeletein. A fő komponensek:

- Egy Spring-alapú webszolgáltatás, amely az adatszeletelést, modell-szerializálást és együttes (ensemble) modellek összeállítását végzi.
- Egy Android alkalmazás, amely a Deeplearning4J használatával neurális hálózatokat tanít és HTTP-n keresztül kommunikál a szerverrel.
- Egy bájt-tömb alapú adatátviteli protokoll, amely biztosítja a kompatibilitást és az adatintegritást.

A kísérleti eredmények igazolják, hogy bár a mobil eszközök erőforrásai korlátozottak, sikeresen hozzájárulhatnak az ML-modellek tanításához adatpárhuzamos elosztás révén. Amikor az adatszeleteken tanított részmodelleket együttes modellekbe aggregálják, azok hasonló pontosságot érnek el, mint a teljes adathalmazon tanított modellek. Ez a fejezet rávilágít arra, hogy a mobil eszközök kihasználatlan számítási kapacitása értékes lehet az ML-feladatok számára.

A 3. fejezet azt vizsgálja, hogyan lehet a meglévő, Java-alapú ML-könyvtárakat Android környezetben alkalmazni. Mivel a Python nem ideális az Androidhoz, a vizsgálat a Weka, SMILE és Oracle Tribuo könyvtárakra fókuszál. A fejezet dokumentálja e könyvtárak portolási folyamatát Androidra, bemutatva az inkompatibilitási problémákat és az azok megoldásához szükséges módosításokat. Kísérleti benchmark alkalmazás készült, amely összehasonlítja a könyvtárak futási idejét, memóriahasználatát, CPU-terhelését és energiafogyasztását. Főbb megállapítások:

- **A Weka** a leghatékonyabb memória- és energiafelhasználás szempontjából, így általános célú mobil ML-alkalmazásokhoz ideális.
- **A SMILE** bizonyos esetekben a leggyorsabb (pl. K-means klaszterezés), de a legnagyobb CPU-igényű is.
- **A Tribuo** modern API-t kínál és jól bővíthető, azonban nagy adathalmazokon nem mindig teljesít stabilan.

A fejezet megállapítja, hogy bár mindhárom könyvtár használható Androidon bizonyos korlátokkal, a Weka a legalkalmasabb az általános célú mobil ML-fejlesztésekhez. Ezek az eredmények irányt mutatnak azon fejlesztők számára, akik közvetlenül szeretnék integrálni az ML-t Android alkalmazásokba, külső szolgáltatások használata nélkül.

A 4. fejezet a heterogén ML-modellek skálázható webes rendszerekbe való integrálását tárgyalja. Egy mikroszolgáltatás-alapú architektúrát javasol, amely lehetővé teszi különböző környezetekben és nyelveken tanított modellek együttes használatát. Az architektúra jellemzői:

- ML-könyvtárak (pl. Scikit-learn, Weka, TensorFlow) becsomagolása önálló webszolgáltatásokba.
- Két architekturális változat: **direkt modell** (szolgáltatások közvetlen lekérdezése), illetve **gateway modell** (lekérések továbbítása egy aggregátor szolgáltatáson keresztül).
- Dinamikus együttes modellek létrehozása (többségi szavazás, súlyozott szavazás).

Ez a keretrendszer lehetővé teszi a különböző nyelveken írt komponensek együttműködését, támogatja a modellek újrafelhasználását, és hangsúlyozza a mikroszolgáltatások rugalmasságát. A fejezet kitér a kommunikációs költségekre, a hálózati késleltetésre és az architekturális komplexitásra is, ajánlásokat téve ezek mérséklésére.

Az 5. fejezet két innovatív forgatókönyvön keresztül mutat be a gépi tanulás területéhez kapcsolódó alkalmazásokat:

Az első alfejezet egyedülálló kísérletet mutat be, amely azt vizsgálja, hogyan érzékelik és értelmezik a számosságot (az elemek számát) az emberek és a neurális hálózatok. A kísérletek kognitív pszichológiai alapokon nyugszanak, és azt elemzik, hogy a biológiai és mesterséges rendszerek hogyan ismerik fel a kis számú elemeket explicit számlálás nélkül. Több SMNIST adatváltozat került tesztelésre. Az eredmények azt mutatják, hogy a gépek képesek a "subitizing" (gyors becslés) megtanulására, de erősen támaszkodnak a tanító adatokra, és nem érik el az emberi Objektumfájl-rendszer (OFS) természetes hatékonyságát. Ez a vizsgálat új perspektívát ad az ML-modellek kognitív határairól.

A második alfejezet egy játékosított oktatási keretrendszert mutat be, amely programozást tanít Minecraft környezetben a MALMÖ API segítségével. A diákok olyan ügynököket programoznak, amelyek összetett feladatokat hajtanak végre (pl. vörös virágokat gyűjtenek, miközben elkerülik a veszélyeket), miközben megtanulják a döntési fák, a heurisztikák és a megerősítéses tanulás alapjait. A megközelítés növeli a hallgatók motivációját és teljesítményét az alapszintű programozási kurzusokon, és jól példázza, hogyan integrálhatóak az ML-koncepciók játékosított tanulási környezetekbe.

List of Publications

Journal Papers

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.

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.

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.

Conference Proceedings

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

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.

Other papers (not part of the thesis)

P. Jeszenszky, R. Besenczi, M. Szabo, and M. Ispany, “Estimating road traffic flows in macroscopic Markov model,” 2022 IEEE 2nd Conference on Information Technology and Data Science (CITDS). IEEE, pp. 136–141, May 16, 2022. doi: 10.1109/citds54976.2022.9914332.

N. Bátfai, M. Szabó, "Possible neural models to support the design of prime convo assistant." CEUR Workshop Proceedings. Vol. 2874, pp. 46-55, 2021.

N. Bátfai, R. Besenczi, P. Jeszenszky, M. Szabó, and M. Ispány, “Markov modeling of traffic flow in Smart Cities,” Annales Mathematicae et Informaticae, vol. 53, pp. 21–44, 2021, doi: 10.33039/ami.2021.04.008.