REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA
Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie
Département d'informatique

N° d'ordre :SIOD25/M2/2021

Mémoire

Présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : **Systèmes d'information, Optimisation et Décision (SIOD)**

A Machine Learning Approach to QoS Prediction in Microservice Architecture

---

Par :
HACHI FERIAL

Soutenu le 27/06/2022 devant le jury composé de :

BERIMA SALIMA	\	Président
TAREK ZERNADJI	M C B	Rapporteur
HOUHOU OKBA	\	Examinateur

Année universitaire 2021-2022

Acknowledgements

I would like to express my profound and sincere gratitude to my research director, Dr. Zernadji Tarek, for his guidance helped me in all the time of research. My thanks go to all the teachers in the computer department who helped me with my education.

I am very grateful for the love, care, encouragement and prayers of my parents.

My thanks also go to my precious friends especially Oumaima and Mariem, having such great people around me had a huge impact on my life and pushed me forward to always give my best to everyone I love and care about.

Dedication

Praise be to God **(Alhamdulillah)**, who has enabled me to accomplish this humble work.
I'd like to dedicate my work to :
My parents the joy of my life,
Aymen, **Mostapha**, **Morad** and **Mohcen**, my brothers,
Aridj, **Imen** and **Anfel**, my sisters.
Oumaima and **Meriam**, my friends.
To The Batch Of **2021/2022**.

Abstract

Microservices have become enormously popular since traditional monolithic architectures no longer meet the needs of scalability and rapid development cycle. But the dynamically change in this architecture make service discovery mechanisms are required.

The mechanism of service discovery have continuously evolved during the last years to support the effective service composition in microservice applications. Still, the dynamic nature of services are being rarely taken into account for maximizing the desired quality of service. This work proposes using machine learning technique (Attention mechanism), as part of the service discovery process, to maximize Quality of services(QoS).

keywords: Service Discovery, Attention mechanism, QoS, Microservices Architecture.

ملخص:

أصبحت الخدمات المصغرة شائعة بشكل كبير نظرًا لأن البنى التقليدية المتجانسة لم تعد تلبي احتياجات قابلية التوسع ودورة التطوير السريع. لكن التغيير الديناميكي في هذه البنية يجعل آليات اكتشاف الخدمة مطلوبة.

لقد تطورت آلية اكتشاف الخدمة باستمرار خلال السنوات الماضية لدعم تكوين الخدمة الفعال في تطبيقات الخدمات المصغرة. ومع ذلك ، نادرًا ما يتم أخذ الطبيعة الديناميكية للخدمات في الحسبان لتحقيق أقصى قدر من الجودة المرغوبة للخدمة. يقترح هذا العمل استخدام تقنية التعلم الآلي (آلية الانتباه) ، كجزء من عملية اكتشاف الخدمة ، لزيادة جودة الخدمات.

Résumé :

Les microservices sont devenus extrêmement populaires depuis que les architectures monolithiques traditionnelles ne répondent plus aux besoins d'évolutivité et de cycle de développement rapide. Mais l'évolution dynamique de cette architecture rend nécessaires des mécanismes de découverte de service.

Le mécanisme de découverte de services a continuellement évolué au cours des dernières années pour prendre en charge la composition efficace des services dans les applications de microservices. Pourtant, la nature dynamique des services est rarement prise en compte pour maximiser la qualité de service souhaitée. Ce travail propose d'utiliser la technique d'apprentissage automatique (mécanisme d'attention), dans le cadre du processus de découverte de service, pour maximiser la qualité des services.

Contents

General Introduction **11**

1 Microservices Architecture **13**

1.1 Introduction . . . 13
1.2 Definition . . . 13
1.3 Microservices Characteristics . . . 14
1.4 Microservices architecture VS Service-oriented architecture . . . 15
1.5 Microservices Goals . . . 17
1.6 Microservices advantages . . . 18
1.7 Microservices disadvantages . . . 18
1.8 Service discovery in microservices . . . 19
1.8.1 Definition . . . 19
1.8.2 The Client-Side Discovery Pattern . . . 19
1.8.3 The Server-Side Discovery Pattern . . . 21
1.9 Service Discovery work . . . 22
1.10 Conclusion . . . 23

2 Machine Learning **24**

2.1 Inroduction . . . 24
2.2 Machine Learning . . . 24
2.2.1 Supervised Learning . . . 24
2.2.2 Unsupervised Learning . . . 26
2.2.3 Semi-supervised Learning . . . 26
2.2.4 Reinforcement Learning . . . 27
2.3 Deep Learning . . . 27
2.3.1 Artificial Neural Networks . . . 28
2.3.2 Convolutional Neural Networks . . . 29
2.3.3 Recurrent Neural Network . . . 29
2.3.4 Long-Short Term Memory . . . 30
2.3.5 Attention Mechanism . . . 31
2.3.6 Attention Types . . . 32
2.4 Time Series Forecasting with Deep Learning . . . 33

2.4.1 Time Series . . . 33
2.4.2 Time Series Components . . . 34
2.4.3 Time Series Forecasting . . . 35
2.4.4 Time Series Forecasting with Traditional Machine Learning . . . 35
2.4.5 Time Series Forecasting with Deep Learning . . . 35
2.5 Conclusion . . . 38

3 System design **39**
3.1 Introduction . . . 39
3.2 Related Work . . . 39
3.3 System Architecture . . . 40
3.3.1 Machine Learning Process Flow . . . 40
3.4 Deep learning model architecture . . . 40
3.5 Conclusion . . . 41

4 Implementation and Results **42**
4.1 Intoduction . . . 42
4.2 Work Environment and Development Tools . . . 42
4.2.1 Programming language . . . 42
4.2.2 Deep learning and Attention mechanism kit . . . 43
4.2.3 Frameworks and tools . . . 45
4.3 Implementation phases . . . 47
4.3.1 Creating the DL model . . . 47
4.4 Conclusion . . . 54

List of Figures

1.1 Monolotic and Microservices Architecture[42]. . . . 14
1.2 Microservices vs SOA[77]. . . . 17
1.3 Client-Side Discovery Pattern[4]. . . . 20
1.4 Server-Side Discovery Pattern[4]. . . . 21
1.5 Service discovery parts[4]. . . . 22

2.1 Random Forest Classifier architecture[51]. . . . 26
2.2 Supervised Learning and Unsupervised Learning[68]. . . . 26
2.3 Deep Learning[30]. . . . 28
2.4 Artificial neural networks and biological neural networks[76]. . . . 28
2.5 Recurrent Neural Network[40]. . . . 29
2.6 LSTM cell[69]. . . . 30
2.7 Attention types[11]. . . . 32
2.8 A graph showing the Standard Poor (SP) 500 index for the U.S. stock market for 90 trading days starting on March 16 1999[17]. . . . 34

3.1 The ML process . . . 40
3.2 Deep learning model architecture . . . 41

4.1 Python logo[58]. . . . 42
4.2 Python version used in jupyter notebook. . . . 43
4.3 TensorFlow logo[65]. . . . 43
4.4 Keras logo[65]. . . . 43
4.5 Pandas logo[1]. . . . 44
4.6 NumPy logo[31]. . . . 44
4.7 SKlearn logo. . . . 44
4.8 Matplotlib logo. . . . 45
4.9 Seaborn logo. . . . 45
4.10 Anaconda logo. . . . 46
4.11 Anaconda Environment. . . . 46
4.12 Jupyter logo. . . . 47
4.13 Importing Panda and loading dataset. . . . 47
4.14 . . . 47

4.15 Instance Visualization. 47
4.16 Seaborn pairplot. 48
4.17 Response time and instance visualisation. 48
4.18 Seaborn Boxplot. 49
4.19 data informations. 49
4.20 Label Encoding for object to numeric conversion. 50
4.21 New data informations. 51
4.22 Attention Layer. 52
4.23 Attention model with LSTM. 52
4.24 The model summary. 53
4.25 Model training. 53
4.26 Evaluate model. 54

General Introduction

[1-5]

Monolithic software is built to be self-contained, with strongly connected rather than loosely related components or functionalities. For code to be executed or compiled and for software to operate under a monolithic architecture, each component and its associated components must all be present. Single-tiered monolithic apps merge numerous components into a single huge application. As a result, they frequently have enormous codebases that might be difficult to manage over time, and if one software component has to be modified, other parts may need to be rewritten as well, and the entire application must be recompiled and tested. The procedure can be time-consuming, and it can hinder software development teams' agility and speed.

That's why traditional monolithic designs no longer match the needs of scalability and a quick development cycle, hence micro services have exploded in popularity because it offers multiple advantages.

Micro service architecture is a rapidly evolving architectural pattern for developing real-time industrial automation applications. The loosely coupled property of microservices allow the independence between each service , allowing huge, sophisticated applications to be delivered quickly, often, and reliably and making it more resilient, flexible, adaptable, and cost-effective technique for developing various applications that aid in improving the performance and reliability of the clients' business requirements[23].

The success of large organizations (such as Netflix and Amazon) in developing and deploying services serves as a powerful motivator for other businesses to consider making the switch[15].

Despite these benefits, there are certain obstacles and challenges, such as providing services across the network, security and safety concerns, data sharing, data communication, data optimization, and production. Furthermore, due to autoscaling, failures, and upgrades, the set of service instances changes dynamically. As a result, how services discover, connect, and interact with one another is one of the issues in micro service architecture[56].

Consequently, there is a need for complex service discovery procedures whose mechanisms have constantly evolved over the past years (Like Eureka, Synapse, Zookeeper, etc.)[15]Monolithic applications are designed to handle multiple related tasks. They're typically complex applications that encompass several tightly coupled functions.

Still, these mechanisms do not explicitly account for the context and quality of services, which are transient and change over time for a variety of reasons: a service consumer/provider can change its context due to mobility/elasticity, a service provider's QoS profile can change according to day time, and so on. In these circumstances, our approach envisions a novel service discovery process capable of dealing with uncertainty and potential negative consequences caused by frequent variations in service context and QoS profile.

This work proposes the use of Attention mechanism as part of the service discovery process to predict the QoS of services in micro services architectures.

In the first chapter, we will study the basics of the microservice architecture.

In the second chapter, we will illustrate deep learning methods.

In the Third chapter, we will present the conception and Design of our approach.

The fourth chapter about the implementation and results.

Chapter 1

Microservices Architecture

1.1 Introduction

Microservices Architecture is becoming the mainstream services-based integration model and the de-facto standard for services development for enterprise applications. As enterprise applications tend to become complex, demanding on-the-fly scalability and high responsiveness, microservices play a crucial role in fulfilling these criteria.

Enterprises can realize a strategic vision of an API-based, loosely-coupled, scalable and flexible platform architecture with containerized microservices.

In this chapter, we discuss the salient points of the microservice architecture with a focus on some elements to clarify this term.

1.2 Definition

The term "Microservice Architecture" was first used by Dr. Peter Rogers during a conference on cloud computing in 2005,it has sprung up to describe a particular way of designing software applications as suites of deployable ,scaled and tested independently services[66].

Microservices is an architectural and organizational approach to software development in which software is made up of small, self-contained services that communicate over well defined APIs[72].

An application is constructed utilizing independent components that operate each process of the application as a service using a microservices architecture. These services communicate via lightweight APIs and a well-defined interface. Services are created to support business capabilities, and each one serves a single purpose.

Because each service is self-contained, it may be modified, launched, or scaled to fit the needs of certain application functions[42].

The popularity of microservices has recently been on the rise because they can solve many current IT challenges such as increasing speed, scalability of applications and rapid test processes[47].

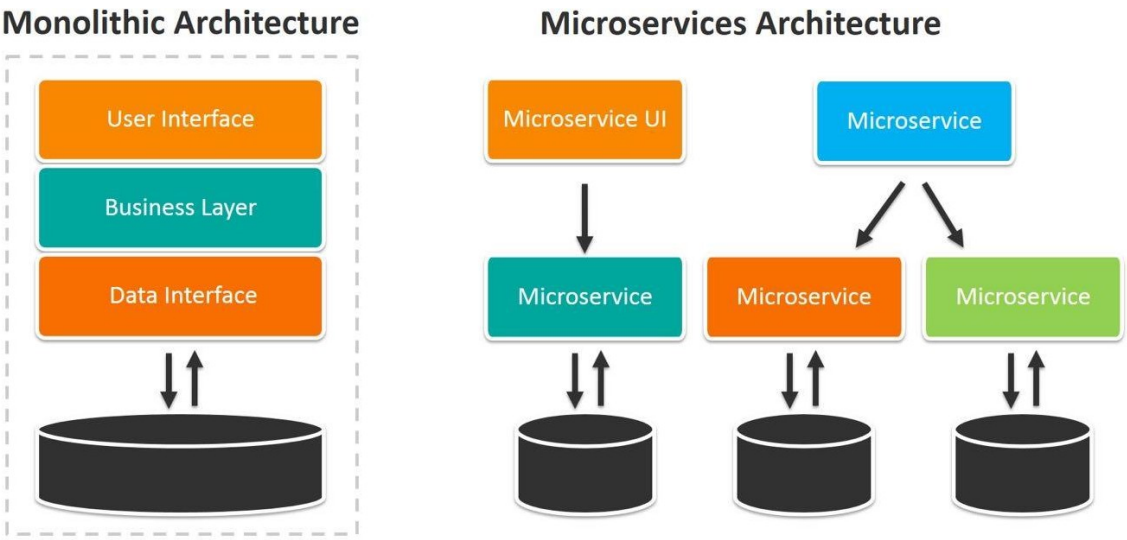


Figure 1.1: Monolotic and Microservices Architecture[42].

1.3 Microservices Characteristics

• Multiple Components:

Software built as microservices canbe broken down into multiple component services. So that each of these services can be deployed, tweaked, and then redeployed independently without compromising the integrity of an application. As a result, you might only need to change one or more distinct services instead of having to redeploy entire applications[12].

• Built For Business:

The microservices style is usually organized around business capabilities and priorities. Unlike a traditional monolithic development approach—where different teams each have a specific focus on, say, UIs, databases, technology layers, or server-side logic—microservice architecture utilizes cross-functional teams[12].

The responsibilities of each team are to make specific products based on one or more individual services communicating via message bus. In microservices, a team owns the product for its lifetime, as in Amazon's oft-quoted maxim "You build it, you run it.

• Simple Routing:

Microservices act somewhat like the classical UNIX system: they receive requests, process them, and generate a response accordingly. This is opposite to how many other products such as ESBs (Enterprise Service Buses) work, where high-tech systems for message routing, choreography, and applying business rules are utilized. You could say that microservices have smart endpoints that process info and apply logic, and dumb pipes through which the info flows.

• Decentralized:

Since microservices involve a variety of technologies and platforms, old-school methods of centralized governance are not optimal. Decentralized governance is favored by the microservices community because its developers strive to produce useful tools that can then be

used by others to solve the same problems. Just like decentralized governance, microservice architecture also favors decentralized data management. Monolithic systems use a single logical database across different applications. In a microservice application, each service usually manages its unique database[12].

• Failure Resistant:

Like a well-rounded child, microservices are designed to cope with failure. Since several unique and diverse services are communicating together, it's quite possible that a service could fail, for one reason or another (e.g., when the supplier is not available). In these instances, the client should allow its neighboring services to function while it bows out in as graceful a manner as possible. However, monitoring microservices can help prevent the risk of a failure. For obvious reasons, this requirement adds more complexity to microservices as compared to monolithic systems architecture[12].

• Evolutionary:

Microservices architecture is an evolutionary design and, again, is ideal for evolutionary systems where you can't fully anticipate the types of devices that may one day be accessing your application.. Many applications start based on monolithic architecture, but as several unforeseen requirements surfaced, can be slowly revamped to microservices that interact over an older monolithic architecture through APIs.

1.4 Microservices architecture VS Service-oriented architecture

Both SOA and microservices architecture are kinds of service architectures and divide the system onto services,because both deal with distributed systems of services communicating over the network and divide the system onto services however in different ways [22][77].

This table below present the difference between Microservices and Service-Oriented architecture[55].

Microservices architecture	Service-oriented architecture
Microservices apps mostly dedicate a database or other type of storage to services that need it.	SOA model has a single data storage layer which shared by all of the services in that application.
Microservices use complex APIs.	Communication between different services in an SOA app uses simple and straight forward approaches.
More focused on decoupling.	Focused on maximizes application service reusability.
Full-stack in nature.	Monolithic in nature
Uses lightweight protocols like HTTP, REST, or Thrift APIs.	Supports multiple message protocols.
It is designed to host services that can function independently.	It is designed to share resources across services.
Quick and easy deployment.	Less flexibility in deployment.
Microservice technology stack could be very large.	The technology stack of SOA is lower compared to Microservice.
A Microservices app could have dozens of services.	An SOA app comprised of two or three services.
They are built to perform a single business task.	SOA applications are built to perform numerous business tasks.
Deployment is straightforward and less time-consuming.	Deployment is a time- consuming process.
In Microservices, systematic change is to create a new service.	A systematic change needed for modifying the monolith.
Emphasis on decoupling.	Focus on maximizing application service reusability.
Deployment is easy and less time-consuming.	The deployment process is time- consuming.

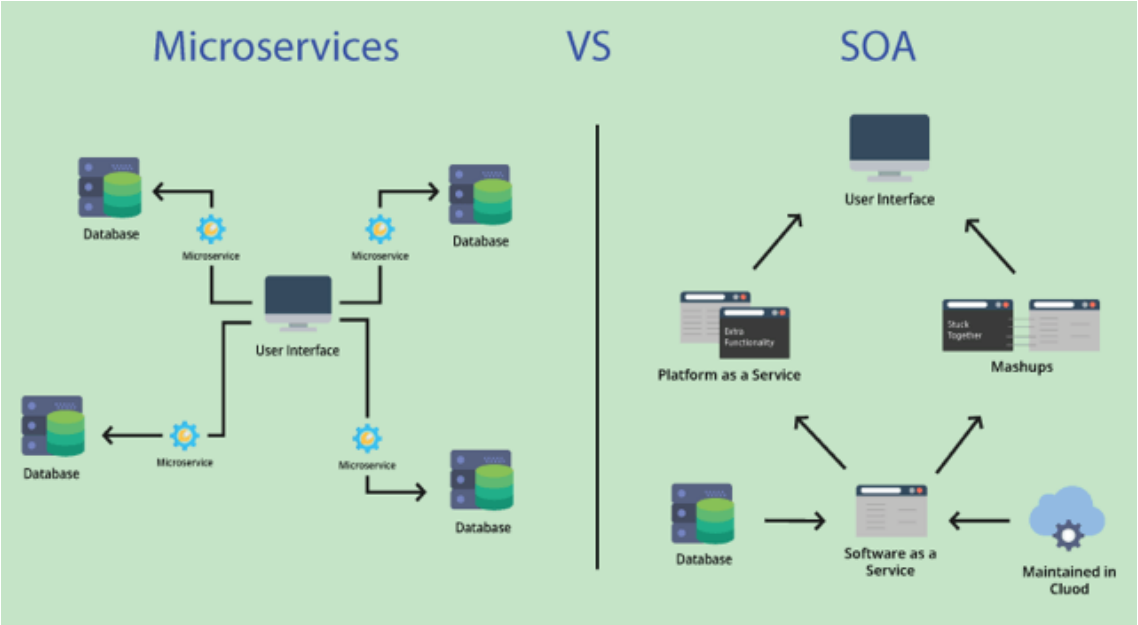


Figure 1.2: Microservices vs SOA[77].

1.5 Microservices Goals

There are a few reasons that contribute to why we need the microservice architecture:

• Continuous delivery:

Microservices offer the ideal architecture for continuous supply. Each program, as well as the environment in which it must execute, is housed in a distinct container with microservices. As a result, each application can be changed in its own container without risk of interfering with other apps. Users will experience minimal downtime, simplified troubleshooting, and no disturbance even if an issue is discovered[32].

• Increase deployment velocity:

Microservice design helps you move at the speed of the market, allowing you to increase deployment velocity and application reliability. Because each application runs in its own containerized environment, it may be moved around without affecting the environment. This reduces the time it takes to get a product to market and improves product reliability[32].

• Empower Developers:

Microservices give developers the tools they need to create higher-quality software. Each component of an application can exist in its own container, controlled and updated independently, with a microservice architecture. This means that instead of having to choose a single less-than-ideal language to use for everything, developers can build applications from numerous components and program each component in the language most suited to its role.

• Reduce cost:

Many industries are seeing rising infrastructure costs as a result of their popular architecture operations. Adding any kind of variation to an application under a monolithic design can be costly because each piece of code in the monolith connects with other components, so a

change in one area affects other features.

• Faster innovation:

Microservices can also help you adapt more quickly to the changing market conditions. Because microservices allow applications to be updated and tested quickly, you can follow market trends and adapt your products faster.

1.6 Microservices advantages

• Agile delivery: Breaking down services into logically modular, self-contained microservices aids Agile delivery, allows for easy integration with the DevOps approach, and reduces time to market [5].

• Ability to use a different technology:developers have the possibility of using many different technologies . This technology diversity is a very common characteristic in applications using microservices and it allows the use of the right tool for the right job[71].

• Easy to understand: Each service is responsible for only one task; therefore, it requires less code. This means that it is easy to understand and has less risk of changes[59][50].

• Scalability: each microservice that contains a given functionality can be independently deployed along with all the dependencies, they can also be independently scaled based on the load [71].

• Fault isolation reduces : if a specific microservices fails, you can isolate that failure to that single service and prevent cascading failures that would cause the app to crash [5][50].

1.7 Microservices disadvantages

• Due to distributed deployment, testing can become complicated and tedious.

• Increasing number of services can result in information barriers.

• The architecture brings additional complexity as the developers have to mitigate fault tolerance, network latency, and deal with a variety of message formats as well as load balancing[50].

• Being a distributed system, it can result in duplication of effort.

• When number of services increases, integration and managing whole products can become complicated.

• In addition to several complexities of monolithic architecture, the developers have to deal with the additional complexity of a distributed system[50].

• Developers have to put additional effort into implementing the mechanism of communication between the services.

• Handling use cases that span more than one service without using distributed transactions is not only tough but also requires communication and cooperation between different teams.

In a microservices application, the set of running service instances changes dynamically.The network locations of instances are assigned dynamically. As a result, a client must employ a service discovery mechanism in order to send a request to a service.

1.8 Service discovery in microservices

1.8.1 Definition

Service discovery is the process by which applications and (micro)services automatically locate each other on a network, eliminating the need for a lengthy configuration setup process. Devices communicate across the network using a standard language, letting devices or services to connect without the need for manual intervention[50].

When migrating to microservices, service discovery is likely the most important piece of infrastructure to implement.

Today, three basic approaches exist to service discovery for microservices[48]:

-The first is to use existing DNS infrastructure.The advantage of this strategy is that DNS(Domain Name System protocol) is already installed in every company. It's also a well-known, highly available distributed system with API implementations in every language imaginable.

-The second is to leverage an existing highly consistent key-value datastore like Apache Zookeeper, Consul, or etcd. These are extremely complex distributed systems. While the initial objective of these systems was not to be used for service discovery, their overall resilience and user-friendly interfaces make them ideal for a variety of service discovery scenarios. Many of the early users of these systems for service discovery did so because it was more convenient - there was a business requirement for a Zookeeper, and then a need for service discovery – therefore the cost of using Zookeeper for service discovery rather than another method was cheap[36].

-Finally, a dedicated service discovery solution like Netflix Eureka is available. This method allows for design choices that are optimal for service discovery. Eureka, for example, places a premium on availability above consistency. However, in order for developers to fully utilize these features, we will need to write some more advanced client libraries, which will increase the technical cost of developing these solutions[36].

There are two main service discovery patterns: client-side discovery and server-side discovery:

1.8.2 The Client-Side Discovery Pattern

In this type of service discovery, the service client or consumer has to search the service registry in order to locate a service provider. Then, the client selects a suitable and free service instance through a load balancing algorithm to make a request[4].

In this pattern, the service instance's location gets registered with the service registry as soon as the service starts. The location information is deleted after the service instance is terminated. This refresh occurs periodically using a heartbeat mechanism.

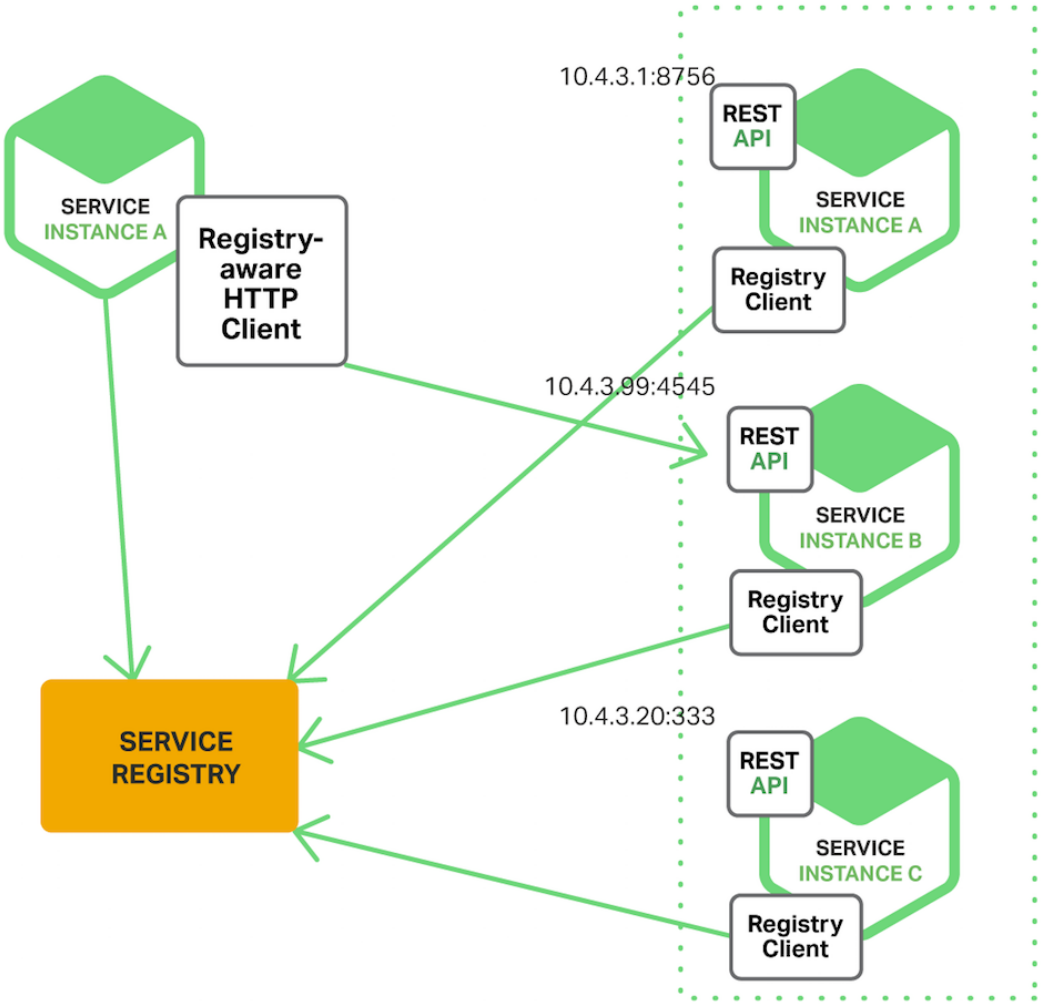


Figure 1.3: Client-Side Discovery Pattern[4].

This type lets you make intelligent decisions regarding load balancing as the consumer remains aware of which service instances are available and capable of taking up the load. It is easy to understand and lets the client determine the network locations of available service instances.

The client's first query is directed at a Discovery Server, which is a central server that acts as a phone book for all service instances that are available. It also provides a layer of abstraction to the service instances[4].

In client-side service discovery, the service might be placed behind an API gateway. If not so, it is the client's responsibility to implement aspects such as authentication, balancing, and cross-cutting.

A popular example of client-side discovery is Netflix OSS, where the service registry is Netflix Eureka. It provides a REST API that manages the integration of service instances and querying available instances. Netflix Ribbon works with Eureka as an inter-process communication (IPC) client to deal with load balancing functions while requests are made to available service instances[4].

1.8.3 The Server-Side Discovery Pattern

In this type of service discovery, the client or consumer does not have to be aware of the service registry. The requests are made through a router, which then searches the service registry itself. When the router finds a valid service instance that is available, it forwards the request and gets the job done[4].

In this pattern, the client does not have to worry about load balancing or finding a suitable service instance. Instead, this job is done by the API gateway, which selects the suitable endpoint for a request coming from the client's side.

Basically, it is the server side's that is responsible for receiving the client server's request and passing it along successfully. This is done by maintaining a registry of service locations and locating the client's desired service without requiring any manual intervention from the consumer side[4].

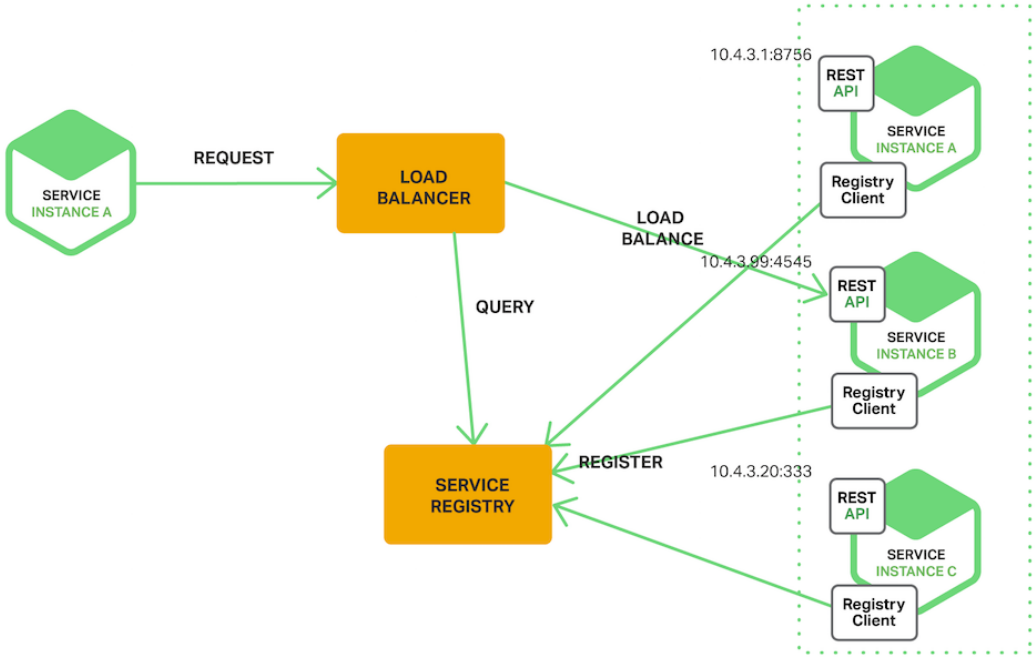


Figure 1.4: Server-Side Discovery Pattern[4].

Each client request is dealt with similarly by the load balancer. Similar to client-side discovery, the service instances are registered with the service registry when the service starts. As soon as the service is terminated, the service instance is deregistered.

A popular example of server-side service discovery is Amazon Web Services (AWS) Elastic Load Balancer (ELB). The ELB is used to balance the load of external traffic from the internet, as well as internal traffic directed to a virtual private cloud (VPC).

The client makes a request through the ELB using its DNS name. The request can be HTTP or TCP. The ELB then performs load balancing of the traffic among Elastic Compute Cloud (EC2) instances or EC2 container service (ECS) containers. The EC2 instances and ECS containers are registered directly with the ELB, without the existence of any separate service registry.

In some deployment environments, like Kubernetes and Marathon, a proxy is run on each host in the cluster. The proxy acts as a server-side load balancer and routes the request using the host's IP address and the port assigned to the service. Then, the request is forwarded to an available service instance running in the cluster[4].

1.9 Service Discovery work

There are three components to Service Discovery:

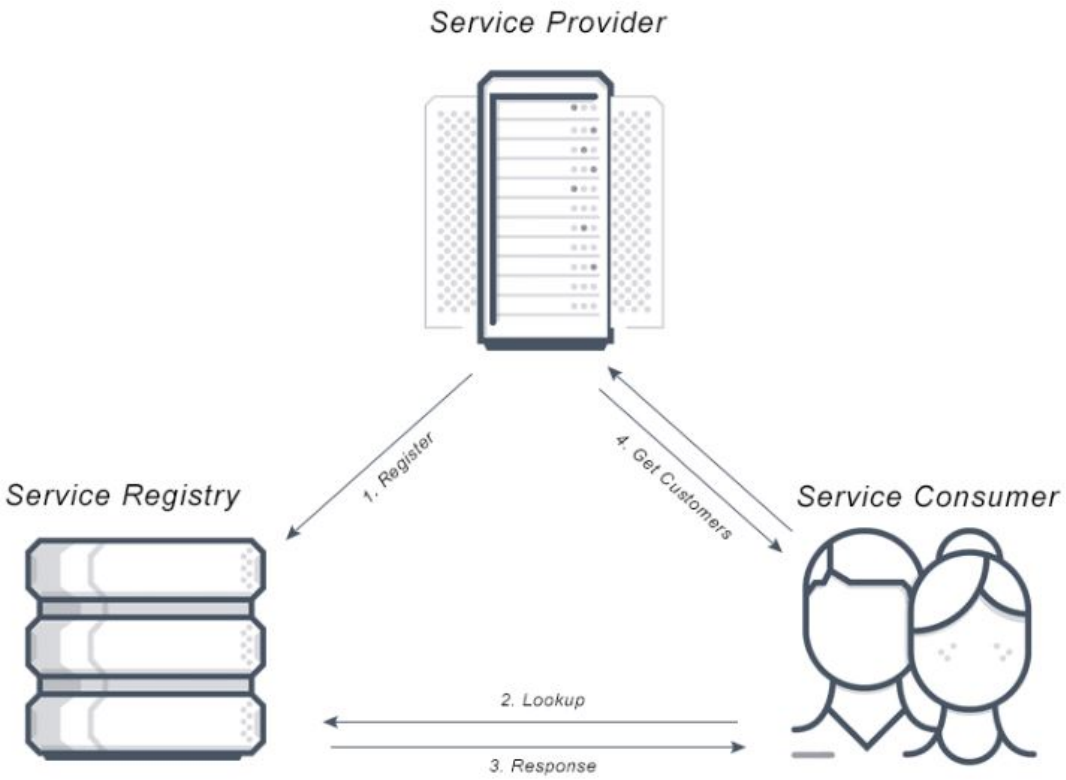


Figure 1.5: Service discovery parts[4].

• The Service Registry: is a key part of service discovery,is a database that contains the network locations of service instances. The service registry needs to be highly available and up to date so clients can go through network locations obtained from the service registry. A service registry consists of a cluster of servers that use a replication protocol to maintain consistency[45].

• The Service Provider: registers itself with the service registry when it enters the system and de-registers itself when it leaves the system.

• The Service Consumer: gets the location of a provider from the service registry, and then connects it to the service provider.

1.10 Conclusion

A microservices architecture is commonly fit for large-scale distributed systems.

Microservices architecture is more comfortable to create and manage autonomous microservices, but it is a difficult task to control on demands of additional network management. Several studies addressed platforms, approaches, and tools which are related to cloud computing applications, internet things, DevOps, real-time applications, and artificial intelligence to adapt to this architectural pattern. And a Service discovery is also the first piece of infrastructure that you should adopt when moving to microservices. In this chapter, we have presented the microservices architecture and the importance of discovery mechanism which rarely takes into account for maximizing the desired quality of service, which is the goal of our project. The next chapter is devoted to an overview of the methods and the prediction algorithms used.

Chapter 2

Machine Learning

2.1 Inroduction

Numerous deep learning architectures have been developed to accommodate the diversity of time series datasets across different domains. The main objective of this chapter is to illustrate deep learning and time series forecasting by learning the most common steps and methods of building a system based on both.

2.2 Machine Learning

Machine learning is an area of artificial intelligence (AI) and computer science that focuses on using data and algorithms to mimic the way people learn, with the goal of steadily improving accuracy.Unlike artificial intelligence applications, machine learning involves learning of hidden patterns within the data (data mining) and subsequently using the patterns to classify or predict an event related to the problem[7].

Simply said, intelligent machines need on information to function, and machine learning provides that knowledge.

It is sufficient to state that all machine learning algorithms are artificial intelligence approaches, yet not all AI methods qualify as machine learning algorithms[6].

There are four basic approaches for machine learning:

2.2.1 Supervised Learning

Many supervised learning approaches have found use in the processing of multimedia material, and supervised learning accounts for a lot of research effort in machine learning.

Supervised learning entails learning a mapping between a set of input variables X and an output variable Y and applying this mapping to predict the outputs for unseen data[20].

Supervised learning is the most important methodology in machine learning which is especially crucial in the processing of time series data.

The most famous approaches in supervised learning :

2.2.1.1 Support Vector Machine(SVMs)

Support vector machine (SVM) is classification method that samples hyperplanes which separate between two or multiple classes. Eventually, the hyperplane with the highest margin is retained ("margin" = the minimum distance from sample points to the hyperplane). The sample point(s) that form margin are called support vectors and establish the final SVM model.[20]

2.2.1.2 Bayes classifiers

based on a statistical model (Bayes theorem: calculating posterior probabilities based on the prior probability and the so-called likelihood). A Naive Bayes classifier assumes that all attributes are conditionally independent, thereby, computing the likelihood is simplified to the product of the conditional probabilities of observing individual attributes given a particular class label.[20]

2.2.1.3 Decision Tree

tree like graphs, where nodes in the graph test certain conditions on a particular set of features, and branches split the decision towards the leaf nodes. Leaves represent lowest level in the graph and determine the class labels. Optimal tree are trained by minimizing impurity (gini) — or maximizing information gain.[64]. A population is divided into branch-like segments that form an inverted tree with a root node, internal nodes, and leaf nodes. The method is non-parametric, which means it can handle huge, complex datasets without imposing a complex parametric framework. There are many algorithms used for developing decision trees(CART, ID3 C4.5, CHAID, QUEST..etc).

2.2.1.4 Linear Regression

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used. [73].

2.2.1.5 Random Forest Classifier

Random Forest Classifier is one of the more elaborate variations of the decision trees [51].

It creates a sequence of decision trees based on a randomly organized selection from the training dataset. Then it gathers the information from the other decision trees so that it could decide on the final class of the test object [51].

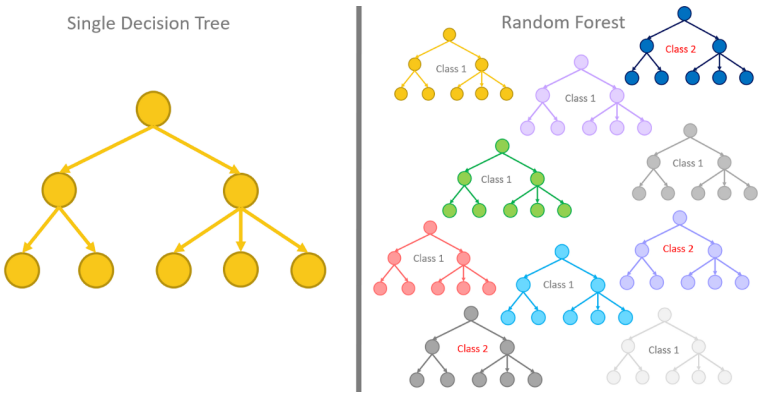


Figure 2.1: Random Forest Classifier architecture[51].

2.2.2 Unsupervised Learning

Unsupervised learning analyzes and clusters unlabeled datasets using machine learning methods. Without the need for human interaction "supervisor", these algorithms uncover hidden patterns or data groupings. It is the best option for exploratory data analysis, cross-selling techniques, consumer segmentation, and picture identification because of its capacity to detect similarities and contrasts in information[26].

The most famous approaches in supervised learning :FP-Growth, K-Means, Fuzzy..etc

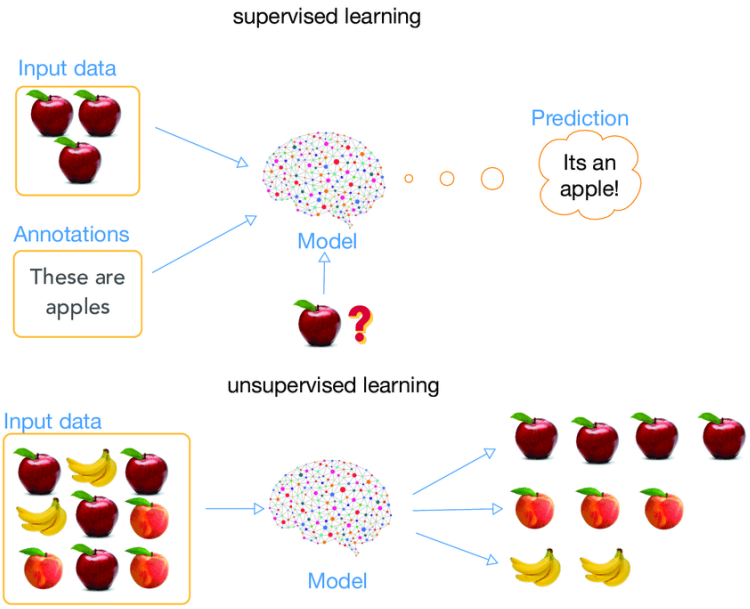


Figure 2.2: Supervised Learning and Unsupervised Learning[68].

2.2.3 Semi-supervised Learning

Semi-supervised learning is a learning paradigm concerned with the study of how computers and natural systems such as humans learn in the presence of both labeled and unlabeled data[75][68].The purpose of semi-supervised learning is to figure out how mixing labeled and unlabeled input affects learning behavior and to create algorithms that take advantage of it.

Semi-supervised learning is of great interest in machine learning and data mining because it can use readily available unlabeled data to improve supervised learning tasks when the labeled data are scarce or expensive[75].

2.2.4 Reinforcement Learning

Reinforcement learning is the training of machine learning models to make a sequence of decisions.In an uncertain, possibly complicated environment, the agent learns to attain a goal. An artificial intelligence meets a game-like circumstance in reinforcement learning. To find a solution to the problem, the computer uses trial and error(Learn from mistakes)[33]. To get the machine to do what the programmer wants, the artificial intelligence gets either rewards or penalties for the actions it performs. Its goal is to maximize the total reward.

2.2.4.1 Common reinforcement learning algorithms

The field of reinforcement learning is made up of several algorithms that take somewhat different approaches. The differences are mainly due to their strategies for exploring their environments.

State-action-reward-state-action (SARSA): This reinforcement learning algorithm starts by giving the agent what's known as a policy. The policy is essentially a probability that tells it the odds of certain actions resulting in rewards, or beneficial states.

Q-learning: This approach to reinforcement learning takes the opposite approach. The agent receives no policy, meaning its exploration of its environment is more self-directed.

Deep Q-Networks: These algorithms utilize neural networks in addition to reinforcement learning techniques. They utilize the self-directed environment exploration of reinforcement learning. Future actions are based on a random sample of past beneficial actions learned by the neural network.

2.3 Deep Learning

Deep learning (DL) is a subset of machine learning (ML) techniques that employ artificial neural networks (ANN) that are inspired by the structure of neurons in the human brain. The term "deep" is a colloquial term that refers to the existence of several layers in an artificial neural network.

DL is a machine learning tsunami in the sense that a small number of innovative approaches have been effectively applied to a wide range of domains (image, text, video, audio, and vision), vastly enhancing prior state-of-the-art results attained over decades. The success of deep learning is also attributable to the increased availability of training material (such as ImageNet for pictures) and the comparatively low cost of GPUs for extremely efficient numerical computation. Google, Microsoft, Amazon, Apple, Facebook, and a slew of other companies use deep learning techniques to analyze massive amounts of data on a daily basis[30].

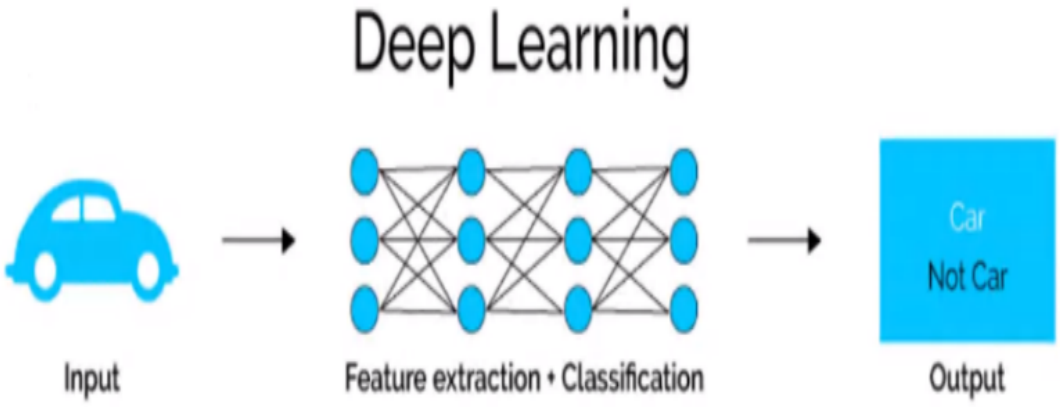


Figure 2.3: Deep Learning[30].

2.3.1 Artificial Neural Networks

The artificial neural network (ANN) is a machine learning method evolved from the idea of simulating the neural networks of human brain[76].

Between the input and output layers, deep neural networks (DNNs) have numerous hidden layers. As a consequence, they may learn features (hidden layers) and provide a classification result from a given input (output layer). They've had a lot of success with part-of-speech tagging, chunking, named entity identification, and semantic role labeling in natural language processing.

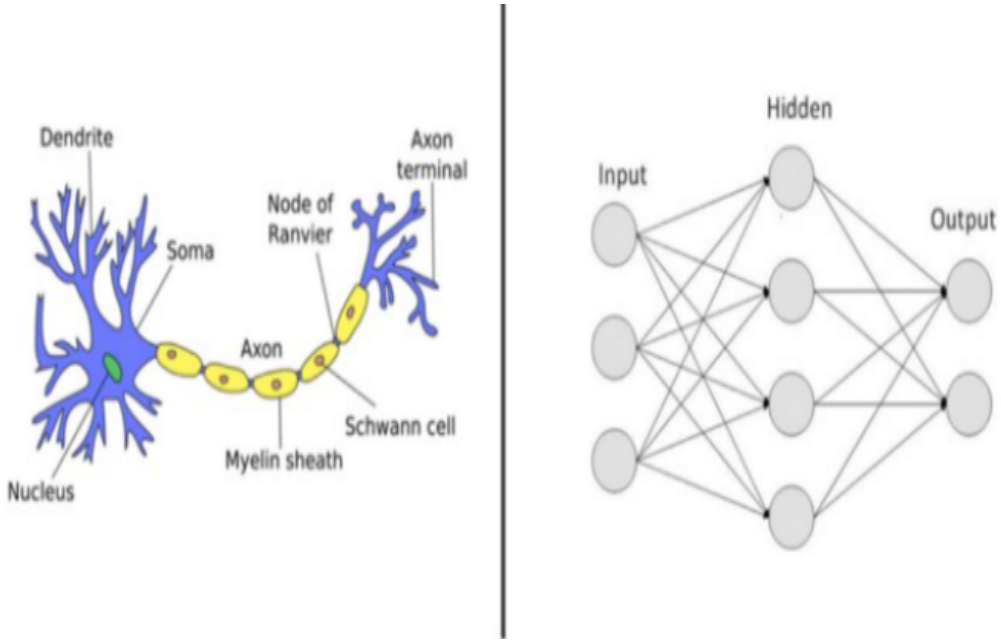


Figure 2.4: Artificial neural networks and biological neural networks[76].

2.3.2 Convolutional Neural Networks

Convolutional networks are a type of neural network that is used to analyze input with a predefined, grid-like architecture.

CNNs have a large range of applications in image and video recognition, recommender systems, image classification, medical image analysis,natural language processing (NLP) and time series forecasting[49].

The name "convolutional neural network" refers to the network's use of the convolutional mathematical procedure. Convolution is a type of linear operation that is specialized. Convolutional networks are simple neural networks with at least one layer that uses convolution instead of ordinary matrix multiplication[29].

2.3.3 Recurrent Neural Network

RNNs are a type of artificial neural network which use sequential data or time series data (sequence model).These deep learning techniques are often employed for ordinal or temporal issues like language translation, natural language processing (nlp), speech recognition and picture captioning, and are used in popular apps like voice search, and Google Translate. Recurrent neural networks, like convolutional neural networks (CNNs), learn from training data. At its core, RNN cells contain an internal memory state which acts as a compact summary of past information[40].

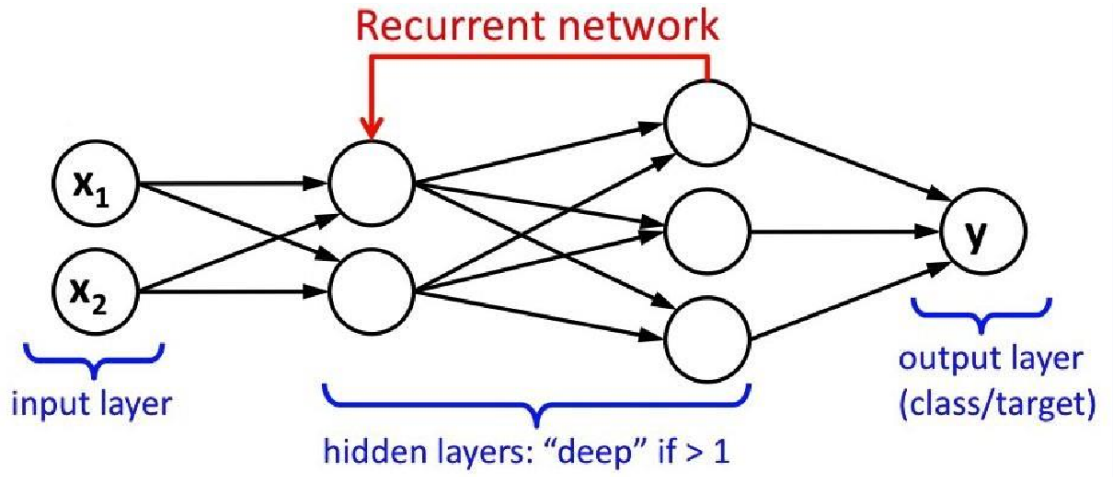


Figure 2.5: Recurrent Neural Network[40].

2.3.3.1 Vanishing Gradient Problem

When propagating the mistake backwards, if the neural network is very deep (has many layers), the gradients (derivatives) measured at the beginning of the prop (last several layers) have a significantly smaller influence on the first few layers. So, in a sense, the gradient diminishes as the network gets farther into the process.

As a result, a basic RNN is unable to detect particularly long-term dependencies between words, and hence fails to detect this subject-verb link when the subject and verb are separated

by a significant distance.This was a major problem in the 1990s and much harder to solve than the exploding gradients. Fortunately, it was solved through the concept of LSTM(Long Short-Term Memory) by Sepp Hochreiter and Juergen Schmidhuber[28].

2.3.4 Long-Short Term Memory

The LSTM model is a recurrent neural system specially designed to overcome the vanishing gradient problems by the multiplicative gates allow LSTM memory cells to store and access information over long periods of time, thereby mitigating the vanishing gradient problem[69][28]. The LSTM architecture consists of a group of recurrently connected subnets, known as memory blocks.These blocks are a differentiable version of a digital computer's memory chips.Each block contains one or more self-connected memory cells and three multiplicative units the input,output and forget gates that provide continuous analogues of write, read and reset operations for the cells.

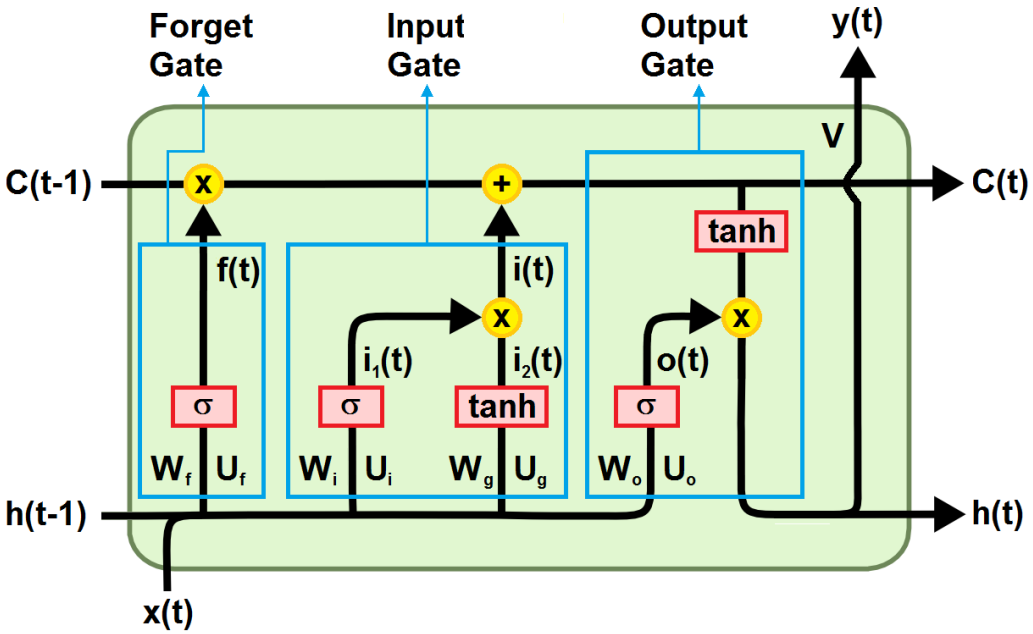


Figure 2.6: LSTM cell[69].

Forget gate : This gate determines whether information should be discarded or saved. The sigmoid function passes information from the previous hidden state as well as information from the current input. The results are between 0 and 1. The closer you go to 0, the more you forget, and the closer you get to 1, the more you keep.

$$f(t) = \sigma\left(x(t)U_f + h(t-1)W_f\right)$$

Input gate : The input gate is used to update the cell state. First, we use a sigmoid

function to combine the prior concealed state and the current input. By changing the values to be between 0 and 1, this determines which values will be updated. A value of 0 indicates that it is not significant, whereas a value of 1 indicates that it is important. To assist control the network, you also send the hidden state and current input into the tanh function to compress values between -1 and 1. The sigmoid result is then multiplied by the tanh output. The sigmoid output will determine which information from the tanh output should be kept.

$$
\begin{aligned}
i_1(t) &= \sigma\left(x(t)U_i + h(t-1)W_i\right) \\
i_2(t) &= \tanh\left(x(t)U_g + h(t-1)W_g\right) \\
i(t) &= i_1(t) * i_2(t)
\end{aligned}
$$

Cell gate : The cell state gets point-wise multiplied by the forget vector.If multiplied by values close to 0, this has the potential to drop values in the cell state. Then we conduct a point-wise addition on the output of the input gate, which changes the cell state to new values that the neural network considers important. As a result, we now have a new cell state.

$$C(t) = \sigma\left(f(t) * C(t-1) + i(t)\right)$$

Output gate : The next hidden state is determined by the output gate. It's important to remember that the concealed state contains data from prior inputs. Predictions are also made using the concealed state. First, we use a sigmoid function to combine the prior concealed state and the current input. The newly adjusted cell state is then sent to the tanh function. To determine what information the hidden state should contain, we multiply the tanh output with the sigmoid output. The concealed state is the output. After that, the new cell state and concealed are carried over to the next time step.

$$
\begin{aligned}
o(t) &= \sigma\left(x(t)U_o + h(t-1)W_o\right) \\
h(t) &= \tanh\left(C_t\right) * o(t)
\end{aligned}
$$

2.3.5 Attention Mechanism

The Attention mechanism is one of the main frontiers in the Deep Learning and is an evolution of the Encoder-Decoder Model, developed in order to improve the performance on long input sequences. Where it have become an integral part of compelling sequence modeling and transduction models in various tasks, allowing modeling of dependencies without regard to their distance in the input or output sequences [11][35][70].In most cases, attention mechanisms are utilized in combination with a recurrent network[52].

The development of attention mechanisms[11][19] has also lead to improvements in long-term dependency learning – with Transformer architectures achieving state-of-the-art performance in multiple natural language processing application[70][21].

The fundamental concept is to allow the decoder to access encoder information selectively during decoding. This is accomplished by creating a unique context vector for each time step of the decoder, calculating it in terms of the previous hidden state as well as all of the encoder's hidden states, and assigning trainable weights to them.

In "encoder-decoder attention" layers, the queries come from the previous decoder layer,and the memory keys and values come from the output of the encoder. This allows every position in the decoder to attend over all positions in the input sequence. This mimics the typical encoder-decoder attention mechanisms in sequence-to sequence models such as[11][70].

As a result, the Attention mechanism allocates varying degrees of relevance to the various pieces of the input sequence, with the more significant inputs receiving greater attention. This explains the model's name.

The attention mechanism can be re-formulated into a general form that can be applied to any sequence-to-sequence task, where the information may not necessarily be related in a sequential fashion.

In NLP models, the attention mechanism produces excellent results since it allows the model to remember all of the words in the input and recognize the most important terms when creating a response.

2.3.6 Attention Types

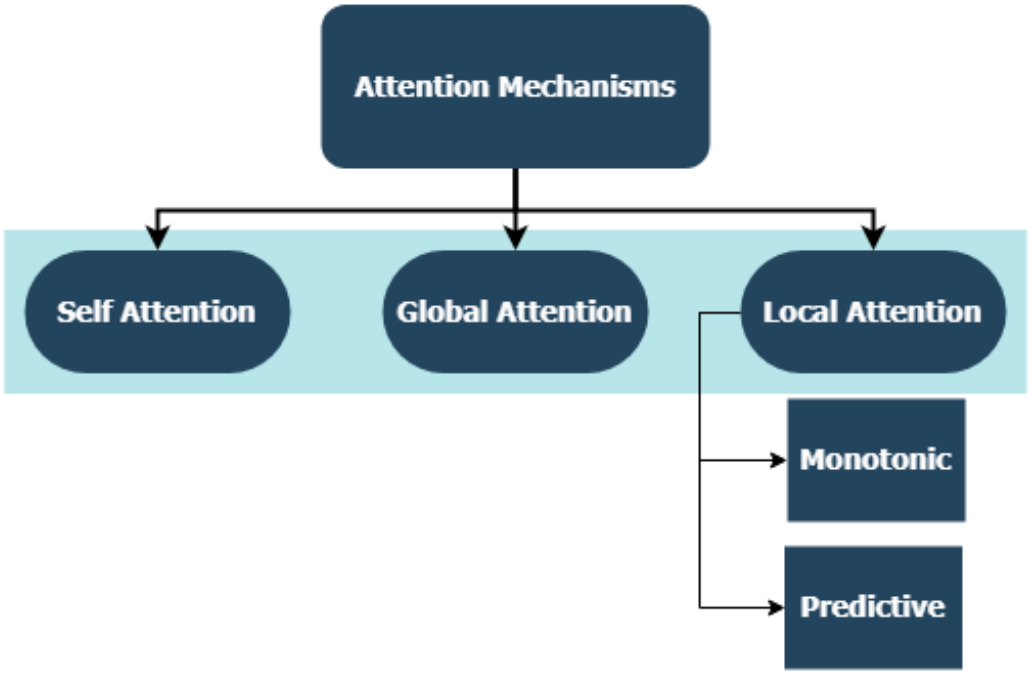


Figure 2.7: Attention types[11].

2.3.6.1 Self Attention

First introduced in Long Short-Term Memory-Networks for Machine ReaOver the past decade, artificial intelligence (AI) technologies have gained increasing importance in a variety of fields. ding by Jianpeng Cheng[70]. The idea is to relate different positions of the same hidden state space derived from the input sequence, based on the argument that multiple components together form the overall semantics of a sequence. This approach brings together these differently positioned information through multiple hops attention. This particular implementation follows A Structured Self-Attentive Sentence Embedding by Zhouhan Lin[62]. where authors propose an additional loss metric for regularization to prevent the redundancy problems of the embedding matrix if the attention mechanism always provides similar annotation weights.

2.3.6.2 Global(Soft) Attention

First introduced in Neural Machine Translation by Jointly Learning to Align and Translate by Dzmitry Bahdanau et al. The idea is to derive a context vector based on all hidden states of the encoder RNN. Hence, it is said that this type of attention attends to the entire input state space.

2.3.6.3 Local(Hard) Attention

First introduced in Show, Attend and Tell: Neural Image Caption Generation with Visual Attention by Kelvin Xu . and adapted to NLP in Effective Approaches to Attention-based Neural Machine Translation by Minh-Thang Luong[67]. The idea is to eliminate the attentive cost of global attention by instead focusing on a small subset of tokens in hidden states set derived from the input sequence.

2.4 Time Series Forecasting with Deep Learning

2.4.1 Time Series

Time series arise as recordings of processes which vary over time. A recording can either be a continuous trace or a set of discrete observations observed over a period of time[53].A Time series is a set of observations taken at specified time; usually at equal intervals. Mathematically a time series is defined by the values Y1,Y2.....Yn of the variable Y at times t1,t2....tn.

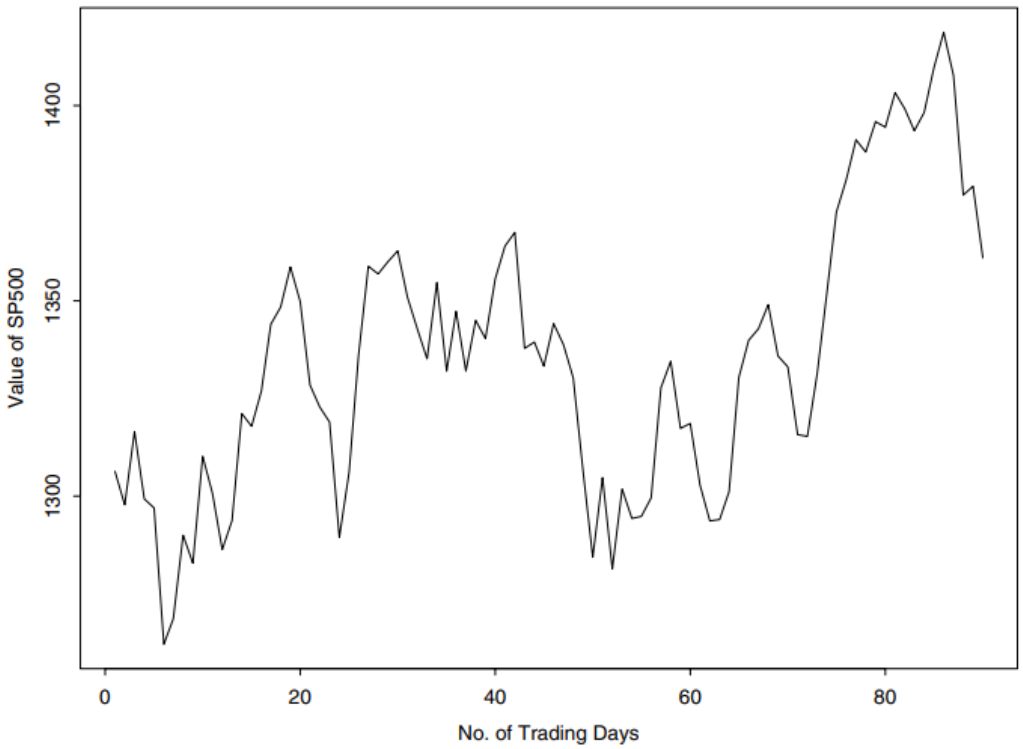


Figure 2.8: A graph showing the Standard Poor (SP) 500 index for the U.S. stock market for 90 trading days starting on March 16 1999[17].

2.4.2 Time Series Components

2.4.2.1 Long term trend

The long-term trend is the data's overall general direction, excluding any short-term impacts such as seasonal changes or noise.

2.4.2.2 Seasonality

Seasonality refers to periodic oscillations that occur repeatedly throughout the course of a time series.

2.4.2.3 Stationarity

The property of stationarity is crucial in time series analysis. If the mean, variance, and covariance of a time series do not fluctuate significantly over time, it is said to be stationary. Many transformations may be used to extract the stationary portion of a non-stationary process.

2.4.2.4 Noise

Every collection of data has noise, which refers to uncontrollable fluctuations or variances.

2.4.2.5 Autocorrelation

The autocorrelation of a time series with a lagged version of itself is used to determine seasonality and trend in time series data.

2.4.3 Time Series Forecasting

Because many various forms of data are kept as time series, time series forecasting has long been a very significant topic of research in many disciplines.For example we can find a lot of time series data in medicine[18][61][74], weather forecasting[14],finance[9], biology[8][63], supply chain management[10] and stock prices forecasting[57][44], etc.

2.4.4 Time Series Forecasting with Traditional Machine Learning

The most classical Machine Learning models used to solve this problem are ARIMA models and exponential smoothing.

ARIMA stands for combination of Autoregressive (AR) and Moving Average (MA) approaches within building a composite model of the time series[13]. This model is really simple, yet it has the potential to provide good outcomes. In order to manage the autocorrelation encoded in the data, it includes parameters to account for seasonality, long-term trend, autoregressive, and moving average terms. Exponential smoothing predictions are based on weighted averages, similar to ARIMA models, but distinct diminishing weights are allocated to each observation, and as we go further away from the present, less significance is given to observations.

Traditional Machine Learning models are generally recognized to have a number of drawbacks:

-Missing values can have a significant impact on model performance.

-They are unable to discern complicated patterns in data.

-They often operate best in short-term predictions rather than long-term forecasts.

2.4.5 Time Series Forecasting with Deep Learning

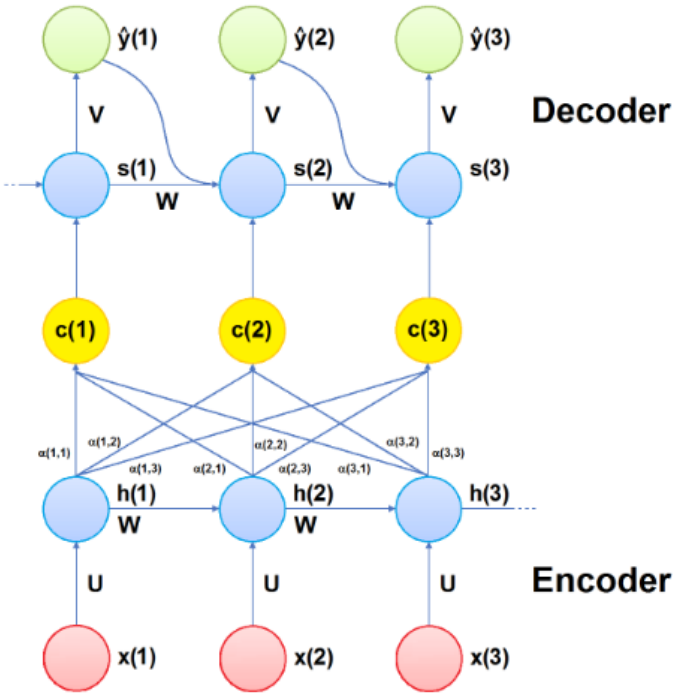
Given the increasing availability of data and computer power in recent years, Deep Learning has become a critical component of the current generation of Time Series Forecasting models, which has shown great results.

Deep learning methods, such as CNNs, RNNs and Long Short-Term Memory Networks, Attention mechanism can be used to automatically learn the temporal dependence structures for challenging time series forecasting problems.

Attention mechanism has the benefit of being more efficient and more interpretable than other Deep Learning models, which are typically regarded as black boxes since they lack the capacity to explain their outputs, thanks to the attention weights.

2.4.5.1 Time Series Forecasting with Attention Mechanism

Recent works has also demonstrated the benefits of using attention mechanisms in time series forecasting applications, with improved performance over comparable recurrent networks [25][38][39]54]. For instance, [25] use attention to aggregate features extracted by RNN encoders, with attention weights produced.



Encoder: The encoder is made up of a stack of recurrent units, which can be RNNs, LSTM cells, or GRU cells. The representation of each input sequence is calculated at each time step as a function of the previous time step's hidden state and the current input. All of the encoded information from the preceding hidden representations and inputs is contained in the final hidden state.

$$h(t) = f(Wh(t-1) + Ux(t))$$

Context vector: The fundamental distinction between the Attention mechanism and the Encoder-Decoder model is that at each time step t of the decoder, a different context vector c(t) is computed.

To calculate the context vector c(t) for time step t, follow these steps. To begin, the so-called alignment scores e(j,t) are generated using the following weighted sum for each combination of encoder time step j and decoder time step t:

$$e(j,t) = V_a \tanh\left(U_a s(t-1) + W_a h(j)\right)$$

Wa, Ua, and Va are trainable weights known as attention weights in this equation. The weights Wa correspond to the encoder's hidden states, the weights Ua to the decoder's hidden states, and the weights Va to the function that calculates the alignment score.

Over the encoder time steps j, the scores e(j,t) are normalized using the softmax function, giving the attention weights (j,t):

$$\alpha(j,t) = \frac{\exp\left(e(j,t)\right)}{\sum_{j=1}^{M} \exp\left(e(j,t)\right)}$$

The relevance of the input of time step j for decoding the output of time step t is captured by the attention weight (j,t). According to the attention weights, the context vector c(t) is constructed as the weighted sum of all the encoder's hidden values:

$$c(t) = \sum_{j=1}^{T} \alpha(j,t) h(j)$$

This context vector enables the input sentence's more relevant inputs to be given greater attention.

Decoder: The decoder now receives the context vector c(t), which computes the probability distribution of the next probable output. This decoding procedure applies to all of the time steps in the input.

The current hidden state s(t) is then computed using the recurrent unit function, using the context vector c(t), the hidden state s(t-1) and the previous time step's output y(t-1) as inputs:

$$s(t) = f\left(s(t-1), \hat{y}(t-1), c(t)\right)$$

The model may thus detect correlations between distinct sections of the input sequence and similar elements of the output sequence using this approach.

The output of the decoder is calculated for each time step by applying the softmax function to the weighted hidden state:

$$\hat{y}(t) = \text{softmax}(Vs(t))$$

2.5 Conclusion

This chapter covered definitions of machine and deep learning and attention mechanism was the main focus of the class due to their importance and also because they are the main topic of this research and some related work. The work design will be introduced in the next chapter, as well as the proposed architectures.

Chapter 3

System design

3.1 Introduction

Service discovery mechanisms have continuously evolved to support the effective service composition in microservice applications. But it could not succeed in taking the dynamic nature into account .Therefore, we propose machine learning technique to take into account this nature and maximize the QoS . In this chapter, we will present the conceptual side of our work, which includes the general architecture as well as the detailed design.

3.2 Related Work

We now review related work on service discovery and the use of machine learning in QoS.

Service Discovery mechanisms have been reviewed extensively in the work of [60]. In general, these mechanisms do not usually take into account QoS concerns such as the response time. However, in a real world invocation environment, aspects such as response time are paramount [43].Ran [54] proposes a model for web service discovery with QoS by extending the UDDI model with the QoS information. Gouscos et al. [27] proposed a simple approach to dynamic Web services discovery that models Web service management attributes such as QoS and price.

ML have been already employed for developing recommendation systems for Web Services, and demonstrated to be effective and efficient. In [46], the authors propose a number of data mining methods to improve service discovery and facilitate the use of Web services. More recently, various ML techniques have been exploited for addressing important aspects related to microservice architectures. In [3], unsupervised learning is used to automatically decompose a monolithic application into a set of microservices. In [24] reinforcement learning has been used for considering QoS factors while assembly services. In [16], bayesian learning and LSTM are used to fingerprint and classify microservices. In [34], reinforcement learning is used to autoscale microservices applications, whereas in [41] random forest regression is used to implement intelligent container scheduling strategy. Finally, in [15] authors used deep neural networks and reinforcement learning to select microservice instances in a given context

and to maximize QoS. We used in our work their exemplar application to generate our dataset and evaluate our approach.

3.3 System Architecture

Our work will be in the service discovery exactly in the service registry that we saw earlier in 1.4. When the service consumer make requests via the Router(Load Balancer), the router queries a service registry to select the proper service instance without taking account for the context and quality of services. In this phase, we apply machine learning technique to maximize the QoS; We extract the QoS data of all the service instances from the service registry to predict the best instance from the QoS parameter (Response Time) of each instance, These parameters represent continuous time series; So it's a time-series forecasting problem. We try to solve using Attention Mechanism.

3.3.1 Machine Learning Process Flow

This figure shows the ML process flow using activity diagram, that make our work more visible.

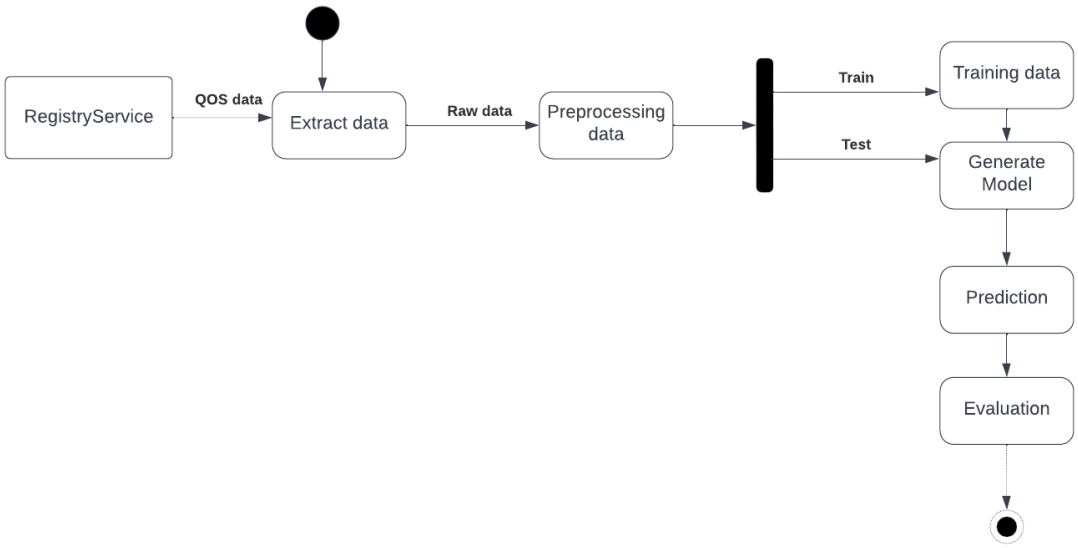


Figure 3.1: The ML process

3.4 Deep learning model architecture

The figure 3.2 represente the DL architecture of our work; we will add a custom Attention layer to LSTM layers.

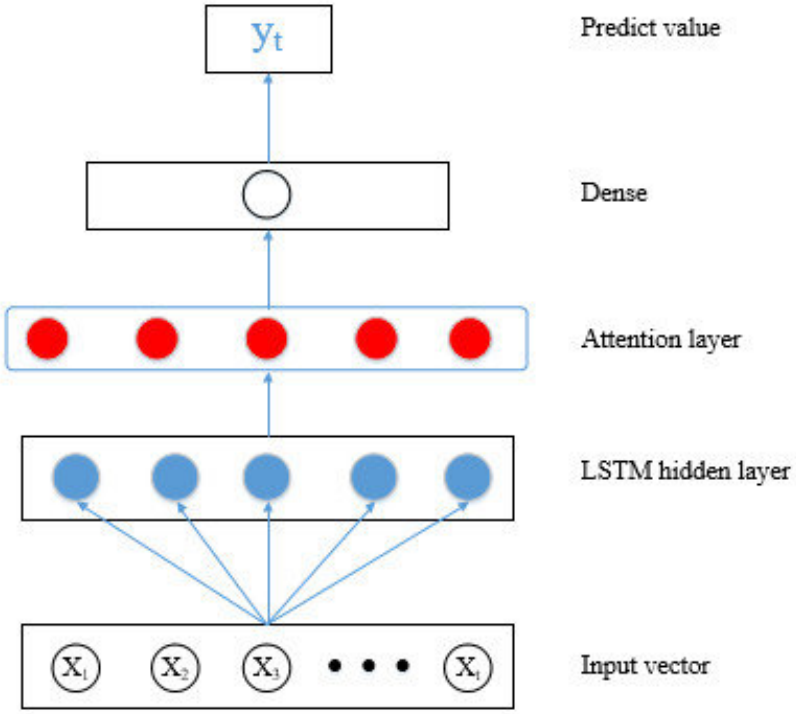


Figure 3.2: Deep learning model architecture

3.5 Conclusion

This chapter described the architecture of our system.The implementation and the results of the experiment,We will present in the next chapter.

Chapter 4

Implementation and Results

4.1 Intoduction

After seeing the design of our work, we come to the implementation.

In this chapter we will present the development environment and the language programming used. Then, we'll go over all of the proposed strategies as well as the outcomes.

4.2 Work Environment and Development Tools

4.2.1 Programming language

4.2.1.1 Python

Python is an easy to understand, versatile programming language. It has efficient high level data structures and a clear but effective approach to object-oriented programming.

Python's elegant syntax and dynamic typing, as well as its interpreted nature, make it an excellent language for scripting and rapid application creation across a wide range of platforms[58].

Figure 4.1: Python logo[58].

```
import sys
sys.version
```

```
'3.9.12 | packaged by conda-forge | (main, Mar 24 2022, 23:17:03) [MSC v.1929 64 bit (AMD64)]'
```

Figure 4.2: Python version used in jupyter notebook.

4.2.2 Deep learning and Attention mechanism kit

The following is the kit of deep learning and Attention mechanism used in our system.

4.2.2.1 TensorFlow

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

Figure 4.3: TensorFlow logo[65].

4.2.2.2 keras

Keras is a high-level neural networks library, written in Python and capable of running on top of either TensorFlow or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research[65].

Figure 4.4: Keras logo[65].

4.2.2.3 Pandas

Pandas is an open source, BSD-licensed library providing high-performance, easy-to use data structures and data analysis tools for the Python programming language[1].

Figure 4.5: Pandas logo[1].

4.2.2.4 NumPy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more[31].

Figure 4.6: NumPy logo[31].

4.2.2.5 Scikit-learn

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

Figure 4.7: SKlearn logo.

4.2.2.6 Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. There is also a procedural "pylab" interface based on a state machine (like OpenGL), designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib.

Figure 4.8: Matplotlib logo.

4.2.2.7 Seaborn

Seaborn is a library for making statistical graphics in Python. It builds on top of matplotlib and integrates closely with pandas data structures. Seaborn helps you explore and understand your data. Its plotting functions operate on data frames and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots.

Figure 4.9: Seaborn logo.

4.2.3 Frameworks and tools

4.2.3.1 Anaconda

Anaconda is a distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment. The distribution includes data-science packages suitable for Windows, Linux, and macOS. It is developed and maintained by Anaconda, Inc., which was founded by Peter Wang and Travis Oliphant in 2012 [2].

Figure 4.10: Anaconda logo.

4.2.3.2 Anaconda Environment

The figure 4.11 represents my environment using in anaconda, I have installed all the libraries that i used to build my code(like TensorFlow, Keras, Pandas.. etc).

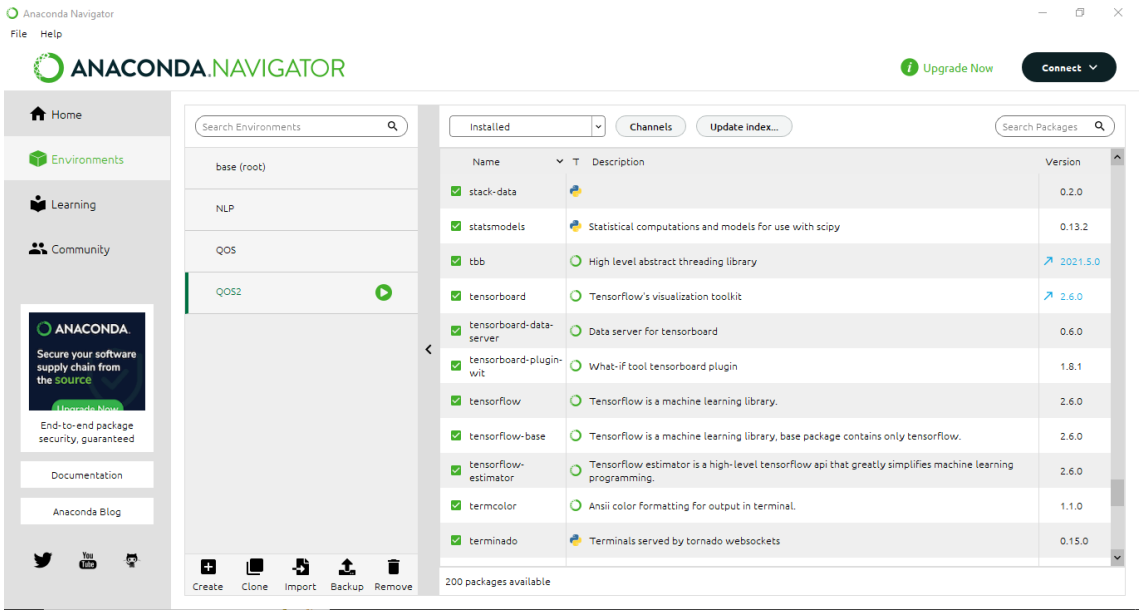


Figure 4.11: Anaconda Environment.

4.2.3.3 Jupyter Notebook

The Jupyter Notebook is the original web application for creating and sharing computational documents. It offers a simple, streamlined, document-centric experience[37].

Figure 4.12: Jupyter logo.

4.3 Implementation phases

4.3.1 Creating the DL model

4.3.1.1 Loading the dataset

To load the data we use panda library.

```
import pandas as pd
data= pd.read_csv(r'C:\Users\PRO\Desktop\Ferial\export_dataframe.csv')
```

Figure 4.13: Importing Panda and loading dataset.

```
data.head()
```

	id	client_context	mean_response_time	Response Time	service_context	Instance	Service Type	status	Timestamp
0	2	C1	0.0	1822	C1	http://192.168.1.12:9030	auth-service	0	2022-05-20 14:22:56.000000
1	3	C1	1822.0	1499	C1	http://192.168.1.12:9030	auth-service	0	2022-05-20 14:23:06.000000
2	4	C1	0.0	9277	C1	http://192.168.1.12:9000	numismatic-service	1	2022-05-20 14:22:51.000000
3	5	C1	9277.0	7199	C1	http://192.168.1.12:9000	numismatic-service	1	2022-05-20 14:23:00.000000
4	6	C1	8238.0	8076	C1	http://192.168.1.12:9000	numismatic-service	1	2022-05-20 14:23:07.000000

Figure 4.14:

4.3.1.2 Visualisation

The figures 4.16 and 4.18 represent the dataset visualisation using matplolib and seaborn libraries for understand well the shape of my dataset and known the content of it.

```
import seaborn as sns
sns.pairplot(data,hue='Instance');
```

Figure 4.15: Instance Visualization.

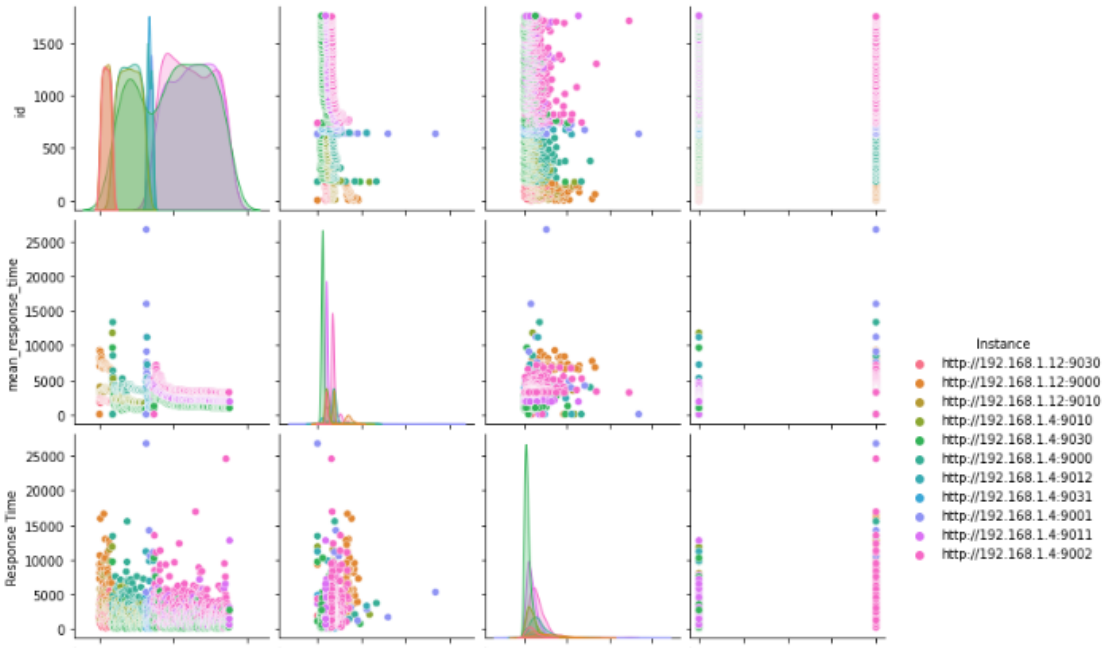


Figure 4.16: Seaborn pairplot.

```
plt.figure(figsize = (10,10))
sns.boxplot(x='Response Time',y='Instance',data=data)
```

Figure 4.17: Response time and instance visualisation.

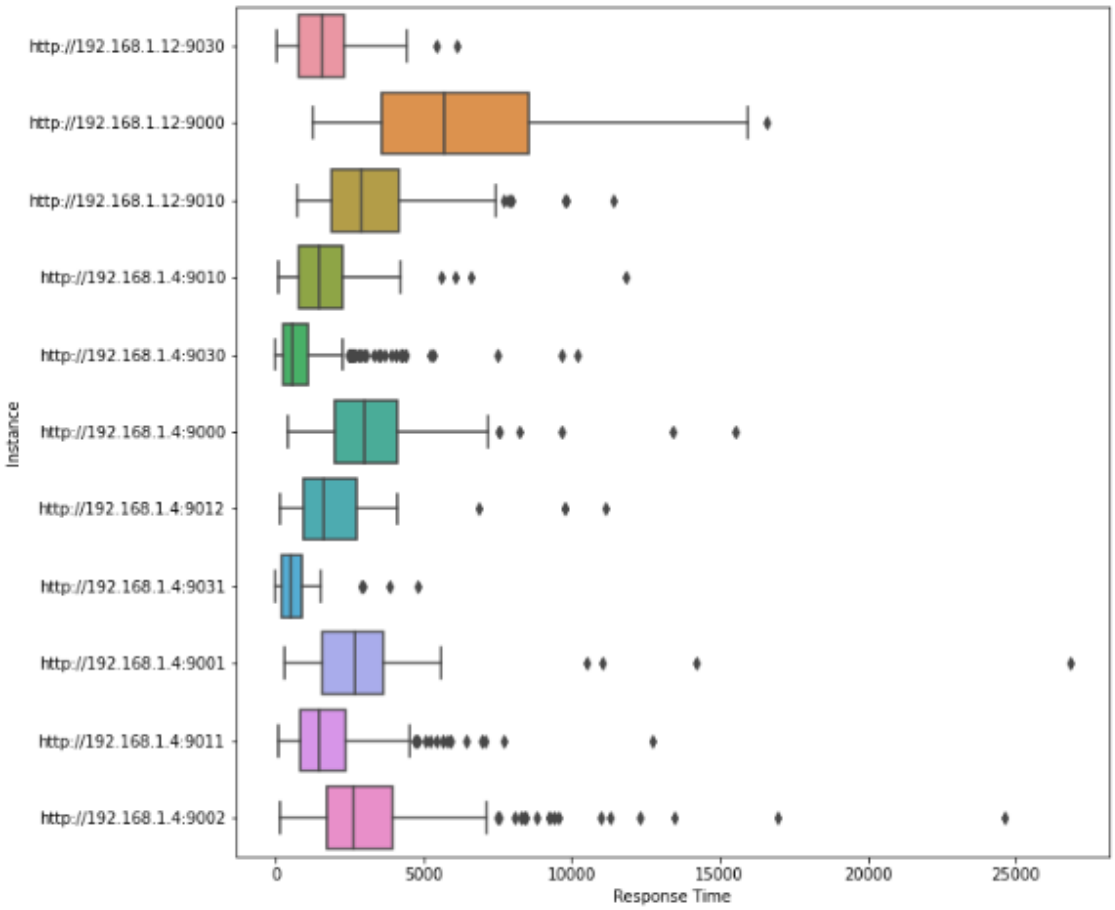


Figure 4.18: Seaborn Boxplot.

4.3.1.3 Feature Engineering

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1756 entries, 0 to 1755
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id                  1756 non-null   int64
 1   client_context      1756 non-null   object
 2   mean_response_time  1756 non-null   float64
 3   Response Time       1756 non-null   int64
 4   service_context     1756 non-null   object
 5   Instance            1756 non-null   object
 6   Service Type        1756 non-null   object
 7   status              1756 non-null   int64
 8   Timestamp           1756 non-null   object
dtypes: float64(1), int64(3), object(5)
memory usage: 123.6+ KB
```

Figure 4.19: data informations.

The figure 4.19 present the function info() which come up with details informations about the type of the data is there a null values or not the name of column, as well as the memory and the memory usage.

It is ovbious that we have different data types (categorical, numerical) in our dataset, which urge us to change the types to use them in prediction phase When of the most used techniques is Label Encoding where all the categorical data get converted to a numerical values where each number represent one label.

However, this part is important to use of the splitting techniques like train test split, shouffle, cross Validation..etc, because splitting the data set into train and test will be a good way to avoid the phenomenon of overfitting.

To convert the above objList features into numeric type, we use a forloop as given below, By using the package of SKLEARN :

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

for feat in objList:
    data[feat] = le.fit_transform(data[feat].astype(str))
```

Figure 4.20: Label Encoding for object to numeric conversion.

The figure 4.21 shows the new data informations after the conversation with label encoding.

```
print (data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1756 entries, 0 to 1755
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id                  1756 non-null   int64
 1   client_context      1756 non-null   int32
 2   mean_response_time  1756 non-null   float64
 3   Response Time       1756 non-null   int64
 4   service_context     1756 non-null   int32
 5   Instance            1756 non-null   int32
 6   Service Type        1756 non-null   int32
 7   status              1756 non-null   int64
 8   Timestamp           1756 non-null   int32
 9   Service_type        1756 non-null   int32
dtypes: float64(1), int32(6), int64(3)
memory usage: 96.2 KB
None
```

Figure 4.21: New data informations.

4.3.1.4 The DL Model

To build our model, we need firstly to create the attention layer.

```
from keras import Model
from keras.layers import Layer
import keras.backend as K

class attention(Layer):
    def __init__(self,**kwargs):
        super(attention,self).__init__(**kwargs)

    def build(self,input_shape):
        self.W=self.add_weight(name='attention_weight', shape=(input_shape[-1],1),
                               initializer='random_normal', trainable=True)
        self.b=self.add_weight(name='attention_bias', shape=(input_shape[1],1),
                               initializer='zeros', trainable=True)
        super(attention, self).build(input_shape)

    def call(self,x):
        # Alignment scores. Pass them through tanh function
        e = K.tanh(K.dot(x,self.W)+self.b)
        # Remove dimension of size 1
        e = K.squeeze(e, axis=-1)
        # Compute the weights
        alpha = K.softmax(e)
        # Reshape to tensorFlow format
        alpha = K.expand_dims(alpha, axis=-1)
        # Compute the context vector
        context = x * alpha
        context = K.sum(context, axis=1)
        return context
```

Figure 4.22: Attention Layer.

build():adding weights.

addweight(): builtin function used to add the weights and the biases .

call(): The call() method implements the mapping of inputs to outputs.it compute the alignment scores, weights, and context.

```
model_att=Sequential()
model_att.add(LSTM(50,return_sequences=True,input_shape=(100,1)))
model_att.add(LSTM(50,return_sequences=True))
attention_layer = model_att.add(attention())
model_att.add(Dense(1))
model_att.compile(loss='mean_squared_error',
                      optimizer='sgd',
                        )
model_att.summary()
```

Figure 4.23: Attention model with LSTM.

loss : Mean squared error (MSE) is the mean overseen data of the squared differences between true and predicted values.

optimizer : Stochastic gradient descent(sgd) used to find the model parameters that correspond to the best fit between predicted and actual outputs.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 100, 50)           10400
_________________________________________________________________
lstm_4 (LSTM)                (None, 100, 50)           20200
_________________________________________________________________
attention (attention)        (None, 50)                150
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 51
=================================================================
Total params: 30,801
Trainable params: 30,801
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.24: The model summary.

The textual summary (Figure 4.24) offers information on the model's layers and their order.

```
model_att.fit(X_train,y_train,validation_data=(X_test,ytest),epochs=10,batch_size=64,verbose=1)
```

```
Epoch 1/10
17/17 [==============================] - 14s 280ms/step - loss: 0.0090 - val_loss: 0.0052
Epoch 2/10
17/17 [==============================] - 3s 176ms/step - loss: 0.0074 - val_loss: 0.0052
Epoch 3/10
17/17 [==============================] - 3s 169ms/step - loss: 0.0073 - val_loss: 0.0054
Epoch 4/10
17/17 [==============================] - 3s 170ms/step - loss: 0.0073 - val_loss: 0.0053
Epoch 5/10
17/17 [==============================] - 3s 173ms/step - loss: 0.0073 - val_loss: 0.0053
Epoch 6/10
17/17 [==============================] - 3s 171ms/step - loss: 0.0073 - val_loss: 0.0054
Epoch 7/10
17/17 [==============================] - 3s 171ms/step - loss: 0.0073 - val_loss: 0.0053
Epoch 8/10
17/17 [==============================] - 3s 171ms/step - loss: 0.0073 - val_loss: 0.0053
Epoch 9/10
17/17 [==============================] - 3s 174ms/step - loss: 0.0073 - val_loss: 0.0054
Epoch 10/10
17/17 [==============================] - 3s 191ms/step - loss: 0.0073 - val_loss: 0.0053
```

Figure 4.25: Model training.

4.3.1.5 Evaluation

```
train_mse_attn = model_att.evaluate(X_train,y_train)
test_mse_attn = model_att.evaluate(X_test, ytest)

print("Train set MSE with attention = ", train_mse_attn)
print("Test set MSE with attention = ", test_mse_attn)
```

```
33/33 [==============================] - 1s 41ms/step - loss: 0.0074
17/17 [==============================] - 1s 40ms/step - loss: 0.0054
Train set MSE with attention =  0.007388617843389511
Test set MSE with attention =  0.0054496587254107
```

Figure 4.26: Evaluate model.

4.4 Conclusion

We have attempted to create ML prediction utilizing Attention mechanism with LSTM throughout the sections of this chapter, demonstrating the technologie employed and their implementation step by step. unfortnuately, we don't satisfied of the resulte because we have a poor dataset.

General Conclusion

With the wide spread of microservices applications ad the evolve of service discovery mechanisms, Even now, these mechanisms do not specifically take service context and quality into consideration.

In this work, we have tried to develop the traditional service discovery mechanism using ML technique (Attention mechanism with deep neural networks) as a part of service discovery process to get the best QoS profile of miroservices instances.

Future study in this area might include, using transfer learning to make the approach more robust .

Bibliography

[1] Pandas.

[2] Anaconda software distribution, 2020.

[3] Muhammad Abdullah, Waheed Iqbal, and Abdelkarim Erradi. Unsupervised learning approach for web application auto-decomposition into microservices. *Journal of Systems and Software*, 151:243–257, 2019.

[4] Omar Al-Debagy and Peter Martinek. A comparative review of microservices and monolithic architectures. In *2018 IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI)*, pages 000149–000154. IEEE, 2018.

[5] Muzaffar Ali, Sabahat Ali, and Atif Jilani. Architecture for microservice based system. a report, 2020.

[6] Mohamed Alloghani, Dhiya Al-Jumeily Obe, Jamila Mustafina, Abir Hussain, and Ahmed Aljaaf. *A Systematic Review on Supervised and Unsupervised Machine Learning Algorithms for Data Science*, pages 3–21. 01 2020.

[7] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[8] Aytaç Altan, Seçkin Karasu, and Stelios Bekiros. Digital currency forecasting with chaotic meta-heuristic bio-inspired signal processing techniques. *Chaos, Solitons & Fractals*, 126:325–336, 2019.

[9] Torben G Andersen, Tim Bollerslev, Peter Christoffersen, and Francis X Diebold. Volatility forecasting, 2005.

[10] Yossi Aviv. A time-series framework for supply-chain inventory management. *Operations Research*, 51(2):210–227, 2003.

[11] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[12] Justus Bogner, Jonas Fritzsch, Stefan Wagner, and Alfred Zimmermann. Microservices in industry: insights into technologies, characteristics, and software quality. In *2019 IEEE international conference on software architecture companion (ICSA-C)*, pages 187–195. IEEE, 2019.

[13] George EP Box and Gwilym M Jenkins. Time series analysis: Forecasting and control san francisco. *Calif: Holden-Day*, 1976.

[14] Sean D Campbell and Francis X Diebold. Weather forecasting for weather derivatives. *Journal of the American Statistical Association*, 100(469):6–16, 2005.

[15] Mauro Caporuscio, Marco De Toma, Henry Muccini, and Karthik Vaidhyanathan. A machine learning approach to service discovery for microservice architectures. In *European Conference on Software Architecture*, pages 66–82. Springer, 2021.

[16] Hyunseok Chang, Murali Kodialam, TV Lakshman, and Sarit Mukherjee. Microservice fingerprinting and classification using machine learning. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pages 1–11. IEEE, 2019.

[17] Chris Chatfield. *Time-series forecasting*. Chapman and Hall/CRC, 2000.

[18] Vinay Kumar Reddy Chimmula and Lei Zhang. Time series forecasting of covid-19 transmission in canada using lstm networks. *Chaos, Solitons & Fractals*, 135:109864, 2020.

[19] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[20] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. Supervised learning. In *Machine learning techniques for multimedia*, pages 21–49. Springer, 2008.

[21] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[22] Shahir Daya, Nguyen Van Duy, Kameswara Eati, Carlos M Ferreira, Dejan Glozic, Vasfi Gucer, Manav Gupta, Sunil Joshi, Valerie Lampkin, Marcelo Martins, et al. *Microservices from theory to practice: creating applications in IBM Bluemix using the microservices approach*. IBM Redbooks, 2016.

[23] Paolo Di Francesco, Ivano Malavolta, and Patricia Lago. Research on architecting microservices: Trends, focus, and potential for industrial adoption. In *2017 IEEE International Conference on Software Architecture (ICSA)*, pages 21–30. IEEE, 2017.

[24] Mirko D'Angelo, Mauro Caporuscio, Vincenzo Grassi, and Raffaela Mirandola. Decentralized learning for self-adaptive qos-aware service assembly. *Future Generation Computer Systems*, 108:210–227, 2020.

[25] Chenyou Fan, Yuze Zhang, Yi Pan, Xiaoyue Li, Chi Zhang, Rong Yuan, Di Wu, Wensheng Wang, Jian Pei, and Heng Huang. Multi-horizon time series forecasting with temporal

attention learning. In *Proceedings of the 25th ACM SIGKDD International conference on knowledge discovery & data mining*, pages 2527–2535, 2019.

[26] Zoubin Ghahramani. Unsupervised learning. In *Summer school on machine learning*, pages 72–112. Springer, 2003.

[27] Dimitris Gouscos, Manolis Kalikakis, and Panagiotis Georgiadis. An approach to modeling web service qos and provision price. In *Fourth International Conference on Web Information Systems Engineering Workshops, 2003. Proceedings.*, pages 121–130. IEEE, 2003.

[28] Alex Graves. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, pages 37–45, 2012.

[29] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern recognition*, 77:354–377, 2018.

[30] Antonio Gulli and Sujit Pal. *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[31] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[32] Pooyan Jamshidi, Claus Pahl, Nabor C Mendonça, James Lewis, and Stefan Tilkov. Microservices: The journey so far and challenges ahead. *IEEE Software*, 35(3):24–35, 2018.

[33] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[34] Abeer Abdel Khaleq and Ilkyeun Ra. Intelligent autoscaling of microservices in the cloud for real-time applications. *IEEE Access*, 9:35464–35476, 2021.

[35] Yoon Kim, Carl Denton, Luong Hoang, and Alexander M Rush. Structured attention networks. *arXiv preprint arXiv:1702.00887*, 2017.

[36] Matthias Klusch. Service discovery., 2014.

[37] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay,

Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press, 2016.

[38] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in Neural Information Processing Systems*, 32, 2019.

[39] Bryan Lim, Sercan O Arik, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *arXiv preprint arXiv:1912.09363*, 2019.

[40] Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A*, 379(2194):20200209, 2021.

[41] Jingze Lv, Mingchang Wei, and Yang Yu. A container scheduling strategy based on machine learning in microservice architecture. In *2019 IEEE International Conference on Services Computing (SCC)*, pages 65–71. IEEE, 2019.

[42] Divyanand Malavalli and Sivakumar Sathappan. Scalable microservice based architecture for enabling dmtf profiles. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 428–432. IEEE, 2015.

[43] Daniel A Menasce. Qos issues in web services. *IEEE internet computing*, 6(6):72–75, 2002.

[44] Prapanna Mondal, Labani Shit, and Saptarsi Goswami. Study of effectiveness of time series modeling (arima) in forecasting stock prices. *International Journal of Computer Science, Engineering and Applications*, 4(2):13, 2014.

[45] Fabrizio Montesi and Janine Weber. Circuit breakers, discovery, and api gateways in microservices. *arXiv preprint arXiv:1609.05830*, 2016.

[46] Richi Nayak and Cindy Tong. Applications of data mining in web services. In *International Conference on Web Information Systems Engineering*, pages 199–205. Springer, 2004.

[47] Rory V O'Connor, Peter Elger, and Paul M Clarke. Continuous software engineering—a microservices architecture perspective. *Journal of Software: Evolution and Process*, 29(11):e1866, 2017.

[48] Organization. Service discovery for microservices.

[49] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[50] Vinod Keshaorao Pachghare. Microservices architecture for cloud computing. *architecture*, 3:4, 2016.

[51] Mahesh Pal. Random forest classifier for remote sensing classification. *International journal of remote sensing*, 26(1):217–222, 2005.

[52] Ankur P Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*, 2016.

[53] Daniel Pena, George C Tiao, and Ruey S Tsay. *A course in time series analysis*, volume 322. John Wiley & Sons, 2011.

[54] Shuping Ran. A model for web services discovery with qos. *ACM Sigecom exchanges*, 4(1):1–10, 2003.

[55] Mark Richards. *Microservices vs. service-oriented architecture*. O'Reilly Media, 2015.

[56] C Richardson. Microservices patterns. shelter island, 2018.

[57] Tae Hyup Roh. Forecasting the volatility of stock price index. *Expert Systems with Applications*, 33(4):916–922, 2007.

[58] Guido van Rossum. Python tutorial: Release 3.6. 4, 2018.

[59] Barakat Saman. Monitoring and analysis of microservices performance. *Journal of Computer Science and Control Systems*, 10(1):19, 2017.

[60] Cristina Schmidt and Manish Parashar. A peer-to-peer approach to web service discovery. *World Wide Web*, 7(2):211–229, 2004.

[61] Sourabh Shastri, Kuljeet Singh, Sachin Kumar, Paramjit Kour, and Vibhakar Mansotra. Time series forecasting of covid-19 using deep learning models: India-usa comparative case study. *Chaos, Solitons & Fractals*, 140:110227, 2020.

[62] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155*, 2018.

[63] Pritpal Singh and Gaurav Dhiman. A hybrid fuzzy time series forecasting model based on granular computing and bio-inspired optimization approaches. *Journal of computational science*, 27:370–385, 2018.

[64] Yan-Yan Song and LU Ying. Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry*, 27(2):130, 2015.

[65] Keras Team. Simple. flexible. powerful.

[66] Johannes Thönes. Microservices. *IEEE software*, 32(1):116–116, 2015.

[67] Uzaymacar. Uzaymacar/attention-mechanisms: Implementations for a family of attention mechanisms, suitable for all kinds of natural language processing tasks and compatible with tensorflow 2.0 and keras.

[68] Jesper E Van Engelen and Holger H Hoos. A survey on semi-supervised learning. *Machine Learning*, 109(2):373–440, 2020.

[69] Greg Van Houdt, Carlos Mosquera, and Gonzalo Nápoles. A review on the long short-term memory model. *Artificial Intelligence Review*, 53(8):5929–5955, 2020.

[70] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[71] Markos Viggiato, Ricardo Terra, Henrique Rocha, Marco Tulio Valente, and Eduardo Figueiredo. Microservices in practice: A survey study. *arXiv preprint arXiv:1808.04836*, 2018.

[72] Hulya Vural, Murat Koyuncu, and Sinem Guney. A systematic literature review on microservices. In *International Conference on Computational Science and Its Applications*, pages 203–217. Springer, 2017.

[73] Sanford Weisberg. *Applied linear regression*, volume 528. John Wiley & Sons, 2005.

[74] Taiyu Zhu, Kezhi Li, Pau Herrero, Jianwei Chen, and Pantelis Georgiou. A deep learning algorithm for personalized blood glucose prediction. In *KHD@ IJCAI*, pages 64–78, 2018.

[75] Xiaojin Zhu and Andrew B Goldberg. Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130, 2009.

[76] Jinming Zou, Yi Han, and Sung-Sau So. Overview of artificial neural networks. *Artificial Neural Networks*, pages 14–22, 2008.

[77] Tom Černý, Michael Donahoo, and Jiri Pechanec. Disambiguation and comparison of soa, microservices and self-contained systems. pages 228–235, 09 2017.