UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

JOÃO ALEXANDRE DA SILVA NETO

DAOS: A drift adaptive system for offloading CEP in Edge Computing

Recife
2022

JOÃO ALEXANDRE DA SILVA NETO

DAOS: A drift adaptive system for offloading CEP in Edge Computing

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Redes de Computadores e Sistemas Distribuídos.

Orientador: Prof. Dr. Kiev Gama
Coorientador: Prof. Dr. Jorge Fonsêca

Recife
2022

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

S586d Silva Neto, João Alexandre da
DAOS: a drift adaptive system for offloading cep in edge computing / João Alexandre da Silva Neto. – 2022.
82 f.: il., fig., tab.

Orientador: Kiev Gama.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn, Ciência da Computação, Recife, 2022.
.
Inclui referências.

1. Redes de Computadores. 2. Aprendizagem de máquina. I. Gama, Kiev (orientador). II. Título.

004.6 CDD (23. ed.) UFPE - CCEN 2022-118

João Alexandre da Silva Neto

“DAOS: A drift adaptive system for offloading CEP in Edge Computing”

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação. Área de Concentração: Redes de Computadores e Sistemas Distribuídos.

Aprovado em: 25/02/2022.

BANCA EXAMINADORA

_____

Prof. Dr. Roberto Souto Maior de Barros
Centro de Informática / UFPE

_____

Prof. Dr. Fernando Antonio Mota Trinta
Departamento de Computação / UFC

_____

Prof. Dr. Kiev Santos da Gama
Centro de Informática / UFPE
(**Orientador**)

ACKNOWLEDGEMENTS

This work is mainly dedicated to my parents, who are fundamental parts of my being and are directly responsible for everything I have achieved so far. It is also dedicated to my grandmother for providing all the support needed so that my brother and I are achieving our goals. In addition, I also dedicate it to my brother for the life partnership and for accompanying me side by side since high school.

My thanks to the supervisors, Prof. Kiev Gama and Prof. Jorge Fonsêca, for all the support given during the development of this work.

I also dedicate this work to my lifelong friends and those I gained at university. My co-workers at C.E.S.A.R were also fundamental for me to balance the work in some complicated moments.

Finally, although this work is not related to health, we live in a pandemic. The thousands of deaths by Covid-19 in Brazil deserve to be remembered. Unfortunately, many of these deaths resulted from a policy of negligence. It is important to emphasize that vaccines save lives.

My thanks to all the healthcare professionals who have worked hard against this virus.

ABSTRACT

Complex Event Processing (CEP) is a paradigm that enables detecting patterns in a stream of events, being widely adopted by use cases such as financial fraud detection and network anomaly detection. Edge computing can extend CEP applications to the edge of the network to deliver a faster response in critical domains. In this scenario, one of the challenges is supporting those applications and keeping optimal resource usage and minimal latency. State-of-the-art solutions have suggested computational offloading techniques to distribute processing between the edge device and a robust cloud instance, reaching that optimization. The traditional offloading techniques use a policy-based approach that compares the device resource usage to predefined thresholds. However, they are few adaptive to changes over time, depending on domain specialists to configure the threshold values. As a solution, decision approaches apply Machine Learning (ML) to learn with the device contextual data to make the best offloading decision. Otherwise, edge devices are known for their resource limitation compared to the cloud, making it hard to use traditional ML models. This scenario demands the usage of online learning algorithms that do not depend on historical data storage and can adapt to changes in the data distribution, known as concept drifts. Therefore, this research proposes DAOS (Drift Adaptive Offloading System), which aims to use online learning and concept drift detection on offloading decisions to optimize the deployment of CEP applications in the edge. Also, it adopts a fallback mechanism to use policies when the models are not reliable. The proposed solution is analyzed through a performance evaluation that compares DAOS with the traditional policy-based mechanism in isolation, varying the CEP application's complexity and data throughput received. The evaluation results show a statistical difference between the approaches, making clear that using online learning and concept drift detection improves CEP offloading decisions and optimizes the resource usage in the edge.

Keywords: edge computing; computation offloading; complex event processing; online machine learning; concept drift.

RESUMO

Processamento de Eventos Complexos (CEP) é um paradigma utilizado para identificar padrões em um fluxo de eventos, viabilizando aplicações para detecção de fraudes financeiras ou anomalias em redes de computadores. Além disso, outro paradigma chamado Computação na Borda é utilizado para estender o CEP e possibilitar que o mesmo seja implantado em dispositivos que ficam mais próximos da origem dos eventos. Consequentemente, isso viabiliza aplicações críticas, onde o tempo de resposta é um fator importante. Um dos desafios nesse cenário é manter essas aplicações sendo executadas na borda com um uso otimizado de recursos e atendendo aos requisitos de tempo de resposta. Para resolver isso, soluções do estado-da-arte sugerem estratégias de transferência de dados computacional para distribuir o processamento entre os dispositivos de borda e uma instância mais robusta na cloud. As técnicas tradicionais de transferência de dados usam um mecanismo baseado em políticas, comparando o uso atual de recursos com limites manualmente especificados. No entanto, essas técnicas são pouco adaptáveis à mudanças ao longo do tempo, exigindo que as políticas sejam constantemente reconfiguradas por especialistas do domínio. Uma solução para isso é utilizar aprendizagem de máquina para aprender com os dados contextuais dos dispositivos e auxiliar o processo de decisão de maneira inteligente. Contudo, os dispositivos de borda possuem limitações de recursos quando comparado com a cloud, dificultando o uso de modelos tradicionais de aprendizagem. Por essa razão, são escolhidos modelos que aprendem de maneira incremental, não dependem de histórico de dados e se adaptam à mudanças de conceito. Portanto, este trabalho propõe a solução DAOS (*Drift Adaptive Offloading System*), que tem como objetivo utilizar aprendizagem online e detecção de mudanças de conceito no processo de tomada de decisão de transferência de dados, visando otimizar a execução de aplicações CEP na borda. Além disso, ele adota um mecanismo de troca de estratégia para utilizar políticas estáticas enquanto os modelos de inteligência não forem confiáveis. Essa proposta é analisada através de uma avaliação de performance que compara o DAOS com a abordagem puramente baseada em políticas, variando a complexidade da aplicação e a taxa de transferência dos dados. A avaliação mostrou que existe uma diferença estatisticamente significativa entre as duas abordagens, evidenciando que as técnicas adotadas pelo DAOS melhoram as decisões de transferência de dados de aplicações CEP na borda.

Palavras-chaves: computação de borda; offloading computacional; processamento de eventos complexos; aprendizagem de máquina; detecção de concept drifts.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ADWIN	ADaptive Sliding WINdow
CEP	Complex Event Processing
DAOS	Drift Adaptive Offloading System
DDoS	Distributed Denial of Service
DSPE	Data Stream Processing Engine
EFDT	Extremely Fast Decision Tree
HT	Hoeffding Tree
IETF	Internet Engineering Task Force
IoT	Internet of Things
JSON	JavaScript Object Notation
kNN	k-Nearest Neighbours
MEC	Mobile Edge Computing
ML	Machine Learning
MQTT	Message Queue Telemetry Transport
NFA	Non-deterministic Finite Automaton
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PMT	Policy Management Tool
PoC	Proof of Concept
PR	Policy Repository
QoS	Quality of Service
SDN	Software-Defined Networking
SQL	Structured Query Language
VFDT	Very Fast Decision Tree
WoT	Web of Things
XML	eXtensible Markup Language

LISTINGS

CONTENTS

1 INTRODUCTION

Big Data is causing a huge transformation in many industries, requiring companies to process a massive amount of data every day. The data handling is now concerned with an accelerating environment described by the 5Vs of Big Data: volume, velocity, variety, veracity, and value (LANEY, 2001). In this scenario, one of the most challenging tasks is how to extract meaningful value from this data, which is vast in terms of volume, type, and sources (BAKSHI, 2012). The cloud computing paradigm provides technologies that can robustly process data. It delivers convenient access to resources that can be quickly provisioned and released with minimal effort (DILLON; WU; CHANG, 2010). The main cloud computing features (elasticity, resources pool, on-demand, self-service, and pay-as-you-go) address the corresponding 5Vs of Big Data (YANG et al., 2017).

Recent trends such as mobile computing and the Internet of Things (IoT) demand a dispersed computing infrastructure instead of relying upon centralized resources provided by the cloud. Its main targets are applications that require a critical response time and the need to avoid the natural latency of processing tasks on the cloud. As a result, it leads to a new paradigm called edge computing in which computing and storage resources are closer to mobile devices or sensors (SATYANARAYANAN, 2017). These resources can be micro data centers, cloudlets, and fog nodes. Most mobile nodes and IoT devices are constrained in terms of CPU, memory, network, and battery, requiring the processing to be offloaded to the cloud when they are not able to carry it out (SHI et al., 2016).

The computation offloading is a strategy that increases these devices' capabilities by relying on more resourceful computers to execute delegated tasks (FLORES et al., 2015) (KUMAR et al., 2013). It is widely adopted in Mobile Edge Computing (MEC) to minimize latency and reduce the response delay of the service, which improves its Quality of Service (QoS) (JIANG et al., 2019). Most of the works try to answer What, When, Where to offload, and Which offload policy will be adopted. To decide *what* to offload and *when* it is worthy, the existing approaches use multiple parameters such as available memory, server speed/load, amount of data, and network metrics (ALAM et al., 2019) (YU; WANG; LANGAR, 2018).

Besides, computation offloading can also optimize the processing of Big Data workloads. With the emergence of IoT and Edge Computing, the data arrives in a continuous stream of events from multiple and distributed sources, hampering its processing. However, this data processing can be done through information processing techniques such as Complex Event Processing (CEP), which detects complex patterns in a data stream by matching incoming events with predefined rules (CUGOLA; MARGARA, 2012). The output of CEP processing is a set of complex and high-level events. The internal state of CEP is often represented via Non-deterministic Finite Automaton (NFA), which can be extracted

from the engine along with its buffers and offloaded to a powerful server (AGRAWAL et al., 2008b).

In the literature, there are approaches that decide *when* to offload CEP workload through a policy-based strategy, which transfers the processing when contextual data of the involved nodes violate resource thresholds (FONSECA; FERRAZ; GAMA, 2018). In these solutions, the values need to be manually optimized to reflect the current offloading requirements, which is a task that depends on domain specialist availability and may not achieve a good optimization considering the varying network conditions and constrained environment.

One alternative is to learn how to decide based on a set of available parameters (e.g. memory and bandwidth usage) from the device and the network. This learning is done through machine learning algorithms, which learn on the fly the optimal policy for deciding *when* to offload the processing (XU; CHEN; REN, 2017) (YU; WANG; LANGAR, 2018). Works in the literature use a variety of algorithms, from reinforcement learning to deep learning (ALAM et al., 2019) (REGO et al., 2017).

Despite having these intelligent techniques for dynamic decisions, edge device limitations demand the usage of memory-efficient algorithms. As the incoming metrics arrive in a stream fashion, there is an opportunity to use online algorithms that fits well because they use one instance at a time, avoiding the need for a large historical data storage (BENCZÚR; KOCSIS; PÁLOVICS, 2019). As a result, the algorithms train incrementally with the most recent data, making it adaptive to changes in data distribution, which are known as concept drifts (GAMA et al., 2014). Those drifts resulted from changes in the distribution of the dataset used to train the models. An adaptive algorithm is essential to guarantee the model's performance will continue acceptable, avoiding unreliable offloading decisions.

This work proposes the Drift Adaptive Offloading System (DAOS) to decide when to offload CEP applications in the edge. It uses online learning and concept drift detection to provide a classifier that can inform when to transfer the processing to cloud. The decision considers metrics from the monitored device, including network conditions and available computation resources. In addition, it proposes a fallback mechanism that uses concept drift occurrences to change the decision approach back to a policy-based strategy, which is more reliable until the online learning model obtains an acceptable performance again.

1.1 RESEARCH PROBLEM

The literature presents strategies for offloading data in many scenarios, including the interaction between edge and cloud environments. There is a recent trend for using machine learning models to improve the offloading decision correctness in that context. These models are trained with the contextual data of edge devices and networking, such as CPU and bandwidth usage. The works that use computation offloading as the infrastructure for CEP applications do not use those intelligent approaches yet, relying on resource thresh-

olds defined by domain specialists. Based on this, the following research questions are set:

- **Q1.** How is it possible to use online learning techniques to offload CEP in the edge in a **fast** and **efficient** way?
- **Q2.** How can the consideration of **concept drift occurrences** increase the effectiveness and reliability of the offloading decision process?

1.2 GOALS

Our main goals are to propose and evaluate a mechanism for detecting when to offload CEP applications' workload and internal state in the edge. It includes an offloading decision algorithm that uses the output of an online learning model, which is trained with device and network contextual data. The evaluation will consider different learning techniques and workloads from a public data set.

The proposal can be separated into a list of specific goals:

1. Design and implement DAOS with the usage of online learning predictions;
2. Define the online learning algorithm that DAOS should use;
3. Evaluate DAOS to answer the research questions (Q1 and Q2).

1.3 RESEARCH STRUCTURE

The rest of this research is organized as follows:

- **Chapter 2 - Background**: Explanation of concepts related to the topics touched by this work.
- **Chapter 3 - Drift Adaptive Offloading System (DAOS)**: Explanation about the proposal of DAOS, including the architecture and its implementation.
- **Chapter 4 - Experiment**: Evaluation of both online learning algorithms and DAOS.
- **Chapter 5 - Related Works**: Discussion about the works that have some relationship with this research.
- **Chapter 6 - Conclusion**: Discussion about the research results and future works.

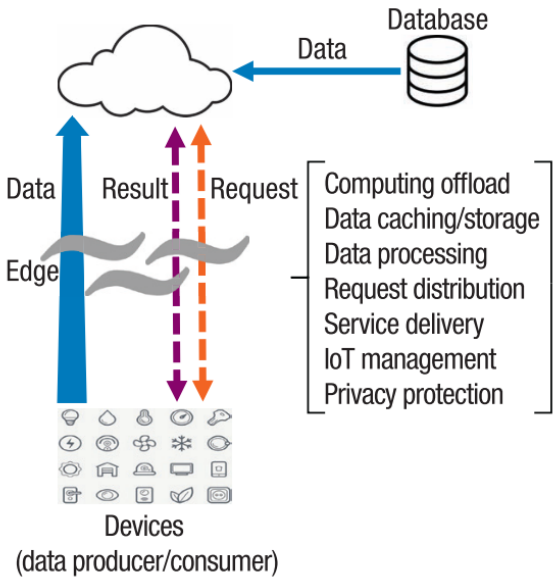
2 BACKGROUND

This chapter explains the main concepts in this work: edge computing, computation offloading, complex event processing, online machine learning, and concept drift. Each section gives the necessary background for understanding the rest of this research.

2.1 EDGE COMPUTING

Cloud computing became a predominant paradigm in the modern world, providing an infrastructure that delivers value to organizations at a low-cost (GONG et al., 2010). Besides the economy factor, this paradigm enables the services to instantly scale by using data centers of cloud providers (HASHEM et al., 2015). In some cases, constrained devices cannot afford to communicate with cloud services directly. To deal with this problem, a new paradigm called Edge Computing has emerged in the past years. It decentralizes the cloud infrastructure, moving computing and storage capabilities to the Internet's edge in proximity with those constrained devices or sensors (SATYANARAYANAN, 2017). Figure 1 gives an overview of the Edge Computing capabilities.

Figure 1 – Overview of Edge Computing Capabilities



Source: (SHI; DUSTDAR, 2016)

Edge computing works as an intermediary in the communication between devices and the cloud. It can perform some computing tasks such as data processing, computing offload, caching, security protection, and device management (SHI; DUSTDAR, 2016). Therefore, instead of relying on the cloud instances directly, the edge devices can receive

a faster response when the processing is executed on the edge layer. Moreover, a lot of systems/tools are being proposed to support the Edge requirements (LIU et al., 2019).

The natural characteristics of Edge Computing lead to an improvement in QoS for end-users. E-commerce platforms can benefit from Edge Computing by offloading shopping-cart updates to the edge and keeping a cache to reduce latency when a user is buying something (SHI; DUSTDAR, 2016). In addition, the edge infrastructure can reduce the impact of cloud outages, performing critical operations during failures. Lastly, there are environments where access to the cloud is a luxury, which is quite common when facing a natural disaster or when the country has a poor networking infrastructure. Both environments can benefit from Edge Computing as well (SATYANARAYANAN, 2017).

2.2 COMPUTATION OFFLOADING

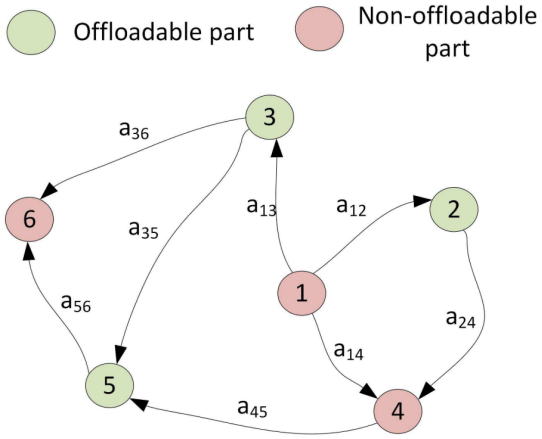
In traditional mobile systems, the execution of computation-intensive tasks is a challenge due to device limitations and network conditions (FORMAN; ZAHORJAN, 1994). To overcome this challenge, mobile agents introduced the idea of task migration between client and server (TCL, 1997). This collaboration is known as computation offloading. According to (KHAN et al., 2014), computation offloading is a technique for migrating computing-intensive tasks from constrained devices to resourceful devices, which increases responsiveness and saves energy. The migration is usually done by detecting computing-intensive portions of application code and delegating its process to remote cloud servers (FLORES et al., 2015).

An important challenge to computation offloading is deciding when the offloading is worthy (MACH; BECVAR, 2017). The decision depends on the system's contextual information, composed of the device, network, and application data. Other questions should be answered before making the decision. According to (JIANG et al., 2019), they are centered on **What** to offload, **When** to offload, **Where** to offload, and **Which** policy will determine the decision strategy. The questions are described below:

- **What:** Defines the type of tasks that should be offloaded. It can be a piece of code or the whole application.
- **When:** Determines the ideal moment to offload the tasks, which depends on the overall context involving metrics of the device, network, and the application state.
- **Where:** Decides the optimal place for executing the offloaded tasks. Will it be in a nearby edge device, in a cloudlet, in a fog node, or in the cloud?
- **Which:** Defines the policies that will drive the offloading decision process, being naturally related to the **When** question.

According to (MACH; BECVAR, 2017), the offloading can be classified into three types: **local** execution, **full** offloading, and **partial** offloading. This classification depends on the application characteristics, which can have **Offloadable** and **Non-offloadable** parts as shown in Figure 2. The workload awareness and dependency between the parts are also important factors to consider in the decision process.

Figure 2 – Relationship between offloadable elements



Source: (MACH; BECVAR, 2017)

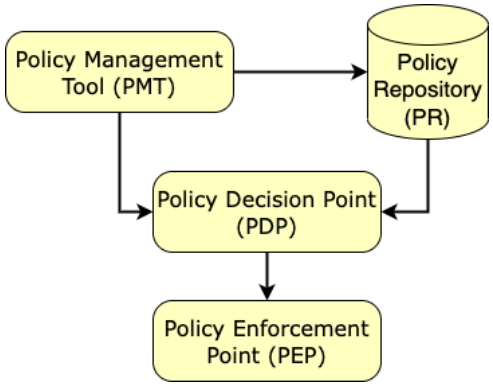
The architecture of systems that coordinate the offloading between devices is composed of three main components: code profiler, system profiler, and decision engine (FLORES et al., 2015). The code profiler determines the portions of code to offload based on computing-intensive attributes. On the other hand, the system profiler collects contextual information about the application surrounding to support the decision. Finally, the decision engine uses the contextual data to choose the appropriate moment to move the processing to a resourceful device.

2.2.1 Policy-based Offloading Mechanism

One of the benefits of implementing computation offloading strategies is making the system dynamically adaptive. This adaptation turns the system into an auto-managed or autonomic one. The policy-based systems use the policies as a mechanism to provide such adaptation, which can indicate how to adjust the system to avoid taking certain states (AGRAWAL et al., 2008a). This work follows the architectural recommendation proposed by the Internet Engineering Task Force (IETF) for implementing policy-based applications, which is illustrated in Figure 3. The main reason for this adoption is reusing widely adopted concepts and references.

In the above architecture, the Policy Management Tool (PMT) is responsible for creating and managing the policies stored in the Policy Repository (PR). The Policy Decision Point (PDP) loads the created policies and makes a decision that is finally enforced by

Figure 3 – Policy Architecture (IETF)



Source: Author

the Policy Enforcement Point (PEP). Generally, the policies are expressed in standard formats such as eXtensible Markup Language (XML) and JavaScript Object Notation (JSON). As an example, the policy defined in the Listing 2.1 is written in XML and defines some rules based on the contextual data about the device involved in the offloading process.

Listing 2.1 – Example of policy that follows the IETF specification

```
<?xml version="1.0" encoding="UTF-8"?>

<policies>
    <policy name="memory" value="70" type="simple"/>
    <policy name="cpu" value="40" type="simple"/>

    <policy type="composed">
        <rule name="cpu" value="60"/>
        <rule name="memory" value="80"/>
    </policy>
</policies>
```

The policies defined in the Listing 2.1 will be considered violated when the memory usage is greater than 70%, or the CPU usage is greater than 40%. It also supports composing policies, which are violated when multiple values are greater than a particular value at the same time.

2.2.2 Machine learning Offloading Mechanism

The *when* question is one of the most challenging steps in offloading decisions since it depends on many parameters from the devices and network. Those parameters are compared against threshold values in policy-based mechanisms to decide when the offloading

should happen. Before this type of mechanism, mobile systems used simple strategies instead of context-sensitive ones. Those strategies use simple timeouts that help to identify delaying tasks and complete them on the cloud. However, the contextual data involved in this decision process usually change over time, being susceptible to many variations out of the control of whoever configured the policy or timeout values. As a result, a dynamic decision-making algorithm becomes necessary (GUO; LIU; LV, 2019).

Nowadays, decision algorithms are using Machine Learning (ML) techniques to learn with new contextual and dynamically adapt to changes. ML algorithms can extract useful information from randomly uncertain, and time-varying data, which is the case in most complex scenarios of offloading in MEC (CAO et al., 2019). On the other hand, the adoption of ML techniques can also increase computation costs and latency. There is vast literature about this adoption, including multiple scenarios and the type of ML algorithms applied (CARVALHO et al., 2020). In this context, (REGO et al., 2017) presented a strategy that uses a decision tree algorithm to help in the offloading decision process, moving the computation to the cloud accordingly.

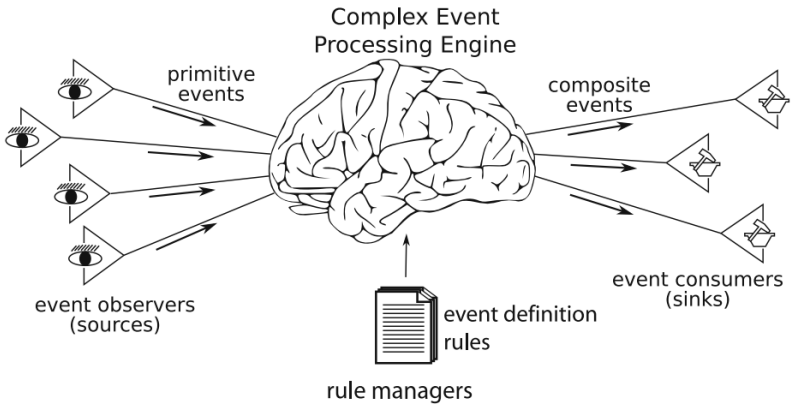
2.3 COMPLEX EVENT PROCESSING

In traditional applications, the data is persistently stored in a database and further used for processing when requested by a user. However, use cases such as anomaly detection require sending alerts when relevant data becomes available. It depends on data that arrives as a timeless information flow, describing the Data Stream Processing Model (DSPM) (BABCOCK et al., 2002). In order to implement this model, the system should expect the data to arrive online from one or more sources, probably out of order and unbounded in size. The output is usually not persisted unless explicitly specified by the users and is delivered as a new data stream.

CEP comes up as an extension of the DSPM, enabling to process the incoming data against a set of defined rules to identify patterns and output new complex events (CUGOLA; MARGARA, 2012). The matching process inherits traditional databases' capabilities, mainly represented through Structured Query Language (SQL) operators such as joins, aggregations, filters, and others. However, traditional databases do not consider detecting patterns that involve sequencing and ordering relations. Many use cases can benefit from CEP capabilities such as fraud detection in financial transactions (ADI et al., 2006) and network anomaly detection (BUTAKOVA et al., 2018).

The publish-subscribe model is one of the foundation bases of CEP. In this system, the data flow as messages coming from multiple publishers (sources) and are delivered to the subscribers (sinks) according to some rules. It introduces a semantic for representing the parts responsible for producing and consuming the events, which is described in the Figure 4 (CUGOLA; MARGARA, 2012). Moreover, the rules for declaring interest in the events can be content-based or defined through topics. Based on this model, CEP systems enable

Figure 4 – Overview of CEP Application



Source:(CUGOLA et al., 2015)

the users to express their interests in composite events, considering the history of received events and the relationship between them. This relationship can be specified with the usage of maps and filters (LUCKHAM; FRASCA, 1998).

The rule managers are responsible for interpreting the external **simple events** and process them to identify **composite events**. In many Data Stream Management Systems (DSMSs), the rules are defined through a high-level language, allowing users to configure the processing information flow as transforming rules. Under the hood, a graph represents this transformation flow, connecting a set of operators. Each operator can receive different events as input and forward them to other operators or directly to the consumers (sinks) (CUGOLA; MARGARA, 2012).

Declarative languages such as SQL have the benefit of helping the users to express complex requirements, promoting the reuse of common logic (DAYARATHNA; PERERA, 2018). As a result, the user can also focus on the transformed outputs rather than on the internal operator's execution flow. The Listing 2.2 provides an example of a CEP query in a declarative language. It calculates the average temperature of ten sensors grouped by id and priority. Esper[1] is one of the event processing systems that supports this type of stream-oriented language to express CEP rules (ETZION; NIBLETT, 2011).

Listing 2.2 – SQL-like Rule

```
select avg(temperature) as aTemperature, id, priority
from Sensor.win:length_batch(10) group by id, priority
```

Alternatively, modern event processing platforms such as Apache Flink can be used to create CEP applications. Flink is a distributed stream processing engine used to process data streams at a large scale. It supports unbounded (stream) and bounded (batch) data sets, delivering real-time analytical insights about them. The applications can be devel-

[1] http://www.espertech.com/esper/

oped on top of the DataStream API, which is adapted to understand declarative languages such as SQL-like queries or expose functions with a pattern-matching semantic. To support the mentioned capabilities, Apache Flink provides the FlinkCEP[2] library, which runs on top of the general-purpose infrastructure. This library solves the challenge of continuously matching events against a set of rules or patterns. The Listing 2.3 demonstrates how an application can be implemented with this library.

Listing 2.3 – DDoS Detection Example

```
val networkSource: DataStream[MqttMessage] = env
  .addSource(new MqttSource(MQTT_BROKER_HOSTNAME,
      APPLICATION_DATA_NETWORK_TOPIC))
  .name("mqtt: network-events-data-source")

...

// DDoS attack pattern (TCP in 1 second > 128)
val start = Pattern.begin[NetworkEvent]("start")
  .where(e => {
    e.protocol == "tcp"
  })
  .within(Time.seconds(1))
  .times(128)

...

mainResults
  .addSink(new MqttSink[ResultEvent](MQTT_BROKER_HOSTNAME,
      APPLICATION_RESPONSE_TOPIC))
  .name("mqtt: network-event-response-sink")
```

The example above shows three main parts of a CEP application written with FlinkCEP:

1. A source component to receive MQTT messages from a particular topic.
2. A CEP pattern for identifying 128 TCP packets within a time window of 1 second.
3. A sink component to deliver the match events to an MQTT topic.

[2] https://ci.apache.org/projects/flink/flink-docs-release-1.13/docs/libs/cep/

2.4 MACHINE LEARNING

Artificial Intelligence (AI) is a technology that can learn with the environment data and perform tasks such as prediction and classification (HAENLEIN; KAPLAN, 2019). In this field, there is a technique called Machine Learning (ML) that aims to improve algorithms by using historical data. Hence, it can perform AI tasks without being explicitly programmed to. It has been used to solve many problems, including image recognition, e-mail filtering, and offloading decisions. The trained algorithms are represented through a model.

In some scenarios, the historical data is not enough for training the model, and the instances arrive almost in real-time. As a solution, online learning algorithms propose to train the model incrementally by using one instance at a time and discarding it when used. According to (BENCZÚR; KOCSIS; PÁLOVICS, 2019), some requirements should be met:

- Online learning updates its model after each data instance without access to all past data, hence the constraints of the data streaming computational model apply.
- Adaptive machine learning models are needed to handle concept drift, which are changes in the data distribution of the models.
- Online learning from big data has to be implemented in a distributed stream processing architecture.

2.4.1 Decision Tree

Most of the state-of-the-art techniques for classification in data streaming are based on decision tree algorithms (BIFET et al., 2017). The decision tree is a supervised learning algorithm that can be used for classification or regression tasks and is widely adopted because of its intuitive interpretation and ability to perform well in large databases (MYLES et al., 2004). The decision tree has different node types: (1) a **root** node without any incoming edges, (2) **internal** nodes that has one incoming edge and many outgoing edges, (3) **leaves** that have one incoming edge and no outgoing edges, also called decision nodes (DATTATREYA, 2009). The internal nodes are responsible for splitting the feature space into two or more sub-spaces.

One challenge when dealing with a huge data set is to select the best internal node to split in the tree. A common solution is to use Information Gain (IG), which is a criterion that calculates how much information a feature has about the class. It uses entropy to measure the impurity or disorder of some examples. This metric indicates the data variance and is calculated through the formula below, where p_i is the probability of randomly picking an element of class i (ROKACH; MAIMON, 2007):

$$E = -\sum_{i}^{C} p_i \log_2 p_i$$

Based on entropy measure, IG represents the quality of a split with the formula. Decision tree algorithms aim to maximize IG and decide what feature will be split or used in the classification process. It can be obtained with the following formula:

$$IG = E(T) - E(T|a)$$

The decision tree algorithms widely used in data mining are ID3, C4.5, and CART. ID3 is a simple algorithm that uses IG as the splitting factor and stops growing the tree when IG is not greater than zero, or all the instances are associated with a single class (ROKACH; MAIMON, 2007). On the other hand, C4.5 is an evolution of ID3 that avoids the tree growing beyond a defined threshold and adopts pruning techniques for removing nodes that do not contribute to the classifier performance. Lastly, the CART algorithm inherits characteristics such as pruning strategies for removing unnecessary leaves. However, the main difference is that it works with binary trees, limiting the internal nodes to two edges. Moreover, CART can generate regression trees to make the leaves predict a number instead of a class (LEWIS, 2000).

2.4.2 Hoeffding Tree (HT)

Decision trees are naturally prepared to perform well in batch settings, where the whole dataset is available. Nevertheless, the data streaming settings demand adjustments to enable them to learn incrementally. In this context (DOMINGOS; HULTEN, 2000) introduced the Hoeffding Tree (HT), whose implementation is called Very Fast Decision Tree (VFDT). Hoeffding tree examines only one example at a time and requires memory for storing only the tree itself. The leaves have the necessary information to allow them to grow. It guarantees a performance similar to the batch algorithms, generating trees of the same quality (DOMINGOS; HULTEN, 2000).

As explained in section 2.4.1, the essential step in a decision tree algorithm is finding the best split possible, directing the examples through different paths. The algorithms iterate over the training dataset in batch settings and compare the attributes IG, selecting the node with the highest value. On the other hand, data streaming cannot perform such a comparison because no training data is available. To overcome this challenge, (DOMINGOS; HULTEN, 2000) adopted the Hoeffding bound as a way to guarantee the algorithm will always decide for the best split possible (HOEFFDING, 1963).

According to the Hoeffding bound, considering a probability of $1-\delta$, the true mean of a random variable will not differ from the estimated mean after n independent observations by more than:

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}$$

In practice, the bound defines a limit for the maximum possible change in the IG difference between two attributes. Therefore, the difference between them in the future will always represent positive separation from the best attribute.

2.4.3 Extremely Fast Decision Tree (EFDT)

An evolution of the Hoeffding Tree (HT) is called Extremely Fast Decision Tree (EFDT) and proposes a Hoeffding Anytime Tree (HATT), obtaining a higher prequential accuracy (MANAPRAGADA; WEBB; SALEHI, 2018). The main difference between both solutions is that HT decides to split only when it finds the best possible split, avoiding revisiting the tree. On the other hand, HATT selects a split as soon as it can be useful, revisiting that decision in the future. Intuitively, HT is more efficient computationally because it often avoids revising the tree. Otherwise, HATT presents better statistical results and guarantees the tree will grow without compromising computational resources.

Despite the adoption of the same Hoeffding bound strategy, HATT uses it to calculate the real advantage of waiting for splitting on the best attribute. In the scenarios where waiting is not a good option, HATT decides to split when the IG due to the top attribute being non-zero. After that, the algorithm will compare the IG between the current top attribute and the split previously decided (MANAPRAGADA; WEBB; SALEHI, 2018). EFDT was not built primarily for self-adaptation, but it has built-in support for adapting when concept drift occurs. The experiments executed (MANAPRAGADA; WEBB; SALEHI, 2018) show EFDT outperforms VFDT on some benchmark datasets. It can be used as an efficient alternative to state-of-the-art techniques.

2.4.4 Naive Bayes

Naive Bayes is considered a simple classifier because it assumes independence between the features, which is a poor assumption in general (RISH et al., 2001). Nonetheless, it usually presents a competitive performance compared with solutions that are not class conditionally independent (WEBB, 2016). The Naive Bayes uses the Bayes's rule for estimating the posterior probability P(Y|x) of each class y given the feature set X, whose formula is described below:

$$P(y|X) = \frac{P(y)P(X|y)}{P(X)}$$

Where $P(X)$ is the same for all classes, being naturally ignored. Moreover, $P(X|y)$ is called class-conditional probability distribution, and $P(y)$ is the prior probability for each class. Both can be easily calculated with a set of categorized features.

Some important aspects of Bayesian classifiers are that they follow a linear model, and the feature array can be incrementally updated. As a result, no special adjustment should be made to use Naive Bayes in data streaming settings. A lot of application domains use Naive Bayes as it proved to be accurate and safe (GUMUS et al., 2014) (SARITAS; YASAR, 2019) (WOOD et al., 2019). In addition, it demands low memory resources and works well with numerical and nominal values. An explicit drawback is that assuming independence between the features, they must be trained with many examples in the training phase.

Another possibility with the Naive Bayes classifier is making a functional enhancement on the leaves of a Hoeffding Tree, refining the classification (GAMA; ROCHA; MEDAS, 2003) (GAMA; MEDAS; ROCHA, 2004). The approach adopted in traditional decision tree algorithms C4.5/CART is that instances with unknown labels are associated with the most frequent class observed during the training. As the leaves maintain the statistics necessary for running Naive Bayes, it can improve the classification results by calculating the probability of each attribute compared to the class labels. This enhancement has been proved to be effective, where the Naive Bayes classification returned a different result from the majority class (BIFET; KIRKBY, 2009).

2.4.5 KNN

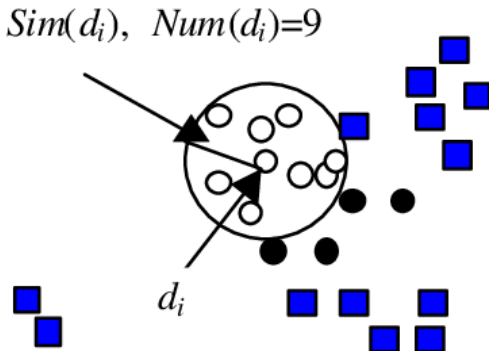
k-Nearest Neighbours (kNN) is a supervised and non-parametric technique that assumes proximity between similar examples (GUO et al., 2003). kNN is usually called lazy because it has no training phase, requiring all the examples during the classification. The similarity between the samples is calculated through a distance measure between them in the feature space. Assuming x is the first point with coordinates $(x_1, x_2, ..., x_p)$ and y is the second point with coordinates $(y_1, y_2, ..., y_p)$, the Euclidian distance can be obtained with the formula below:

$$d(x, y) = \sqrt{\sum_{i=1}^{p}(x_i - y_i)^2}$$

Having the Euclidian distance in hand, another critical variable to consider is the number of neighbors (k) that must be considered when voting the class for that particular example. As a result, the feature space becomes similar to what is illustrated in Figure 5.

Adopting kNN in data streaming scenarios is a challenge because it requires the whole dataset to operate. Therefore, it is hard to be lazy when historical data is unavailable. A possible solution would be setting a sample window and grouping the samples with the closest ones already in memory. Therefore, (LAW; ZANIOLO, 2005) proposed the ANNCAD nearest neighbor algorithm, which has low update cost and no need for fixing the number of neighbors prior to the classification work.

Figure 5 – Close proximity in kNN, considering the distance and the number of neighbors.



Source: (GUO et al., 2003)

2.5 CONCEPT DRIFT

Due to the dynamic environment of data streaming, the data may present unexpected changes over time. These changes affect the underlying statistical properties of the target variable, having a hidden and unforeseen cause, which makes the learning task more complicated (TSYMBAL, 2004). This behavior is defined as concept drift, and when it occurs, the model trained with past data may not be relevant to the new incoming data, leading to poor decisions. It has been proved that concept drift is the root cause of low effectiveness in many data-driven scenarios (LU et al., 2019). An example of concept drift presence is when performing detection of spam e-mails since the strategies of the spammer can change in unpredictable ways, causing the lack of model accuracy.

2.5.1 Formal Definition

The supervised learning problem involves predicting a target variable $y \in R$ given a set of input features $X \in R$, forming a pair of (X, y). In the training data, both X and y are known previously. After being trained, the predictive model is used in a scenario where X is known but y is not known. Concerning the Bayesian Decision Theory (BERGER, 1985), a classification task is defined with the prior probabilities of the categories $P(y)$ and the conditional probability density function $P(X|y)$ for all categories $y = 1, ..., n$ where n is the number of categories. These probabilities can be used to calculate the posterior ones of the categories according to the following formula:

$$P(y|X) = \frac{P(y)P(X|y)}{P(X)}$$

with $P(X)$ being:

$$P(X) = \sum_{y=1}^{c} P(y)P(X|y)$$

Based on classification formal definition, the concept drift can be described as a difference between two time points *t0* and *t1*:

$$\exists X : p_{t0}(X, y) \neq p_{t1}(X, y)$$

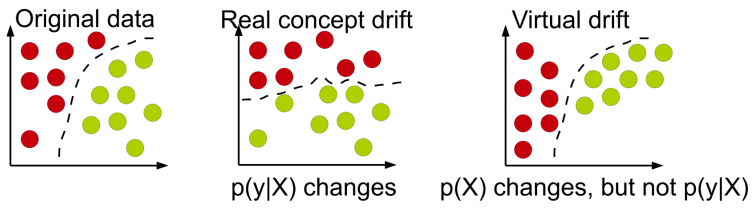
where p_{t0} stands for the joint distribution at time t_0 between the set of examples X and the target variable y. The drift happens when there is a change in the components of the Bayesian equation, which can be the prior probabilities of categories P(y) and class conditional probabilities P(X|y).

2.5.2 Drift Types

As shown in Figure 6, there are two types of concept drifts:

1. Real drift: it happens when there is a change in the posterior probabilities $P(y|X)$ even without a mandatory change in $P(X)$.
2. Virtual drift: it occurs if nothing changes in the posterior probabilities $P(y|X)$, but there are changes in the data distribution $P(X)$.

Figure 6 – Types of concept drift based on the statistical changes



Source:(GAMA et al., 2014)

Figure 6 shows that when there is a real concept drift, the boundary that determines the color changes, influencing the classification. On the other hand, the virtual concept drift does not change the distribution form but the position of the instances.

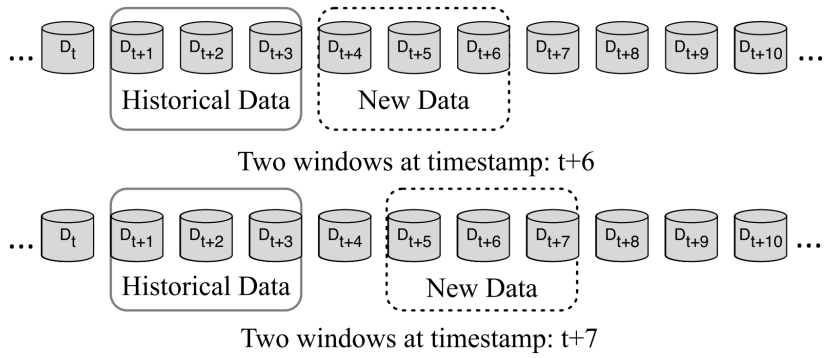
2.5.3 Drift Detection

Concept drift detection uses techniques and mechanisms that identify changes in the underlying distribution, searching for change points in time. (GAMA et al., 2014) discusses four different detection strategies: sequential analysis, control charts, differences between two distributions, and heuristic/contextual based. Firstly, the sequential analysis compares the distribution of two parts of the dataset and assesses the difference through a hypothesis test. Secondly, the control charts approach uses a standard statistical technique widely used to control the manufacturing quality. Another detection strategy is comparing two different time windows, one fixed and the other that slides over time to compare

their distribution. Lastly, the contextual strategy presents a meta-learning technique for identifying intervals with hidden concepts, refining the current concepts accordingly.

In the context of time window approaches, ADaptive Sliding WINdow (ADWIN) is a change detector that uses a two windows (see Figure 7) detection strategy for comparing different data distribution and deciding when a drift occurred (BIFET; GAVALDÀ, 2007). It grows the window with new instances until detecting a concept drift so that it can shrink the window by removing old instances (GRULICH et al., 2018). One additional characteristic of this detector is that it does not require the user to define the size of the compared windows in advance. A change is determined when it rejects the null hypothesis that the distribution of both windows is equal. In this case, the ADWIN compares the difference between the average of the samples in both windows (LIU, 2018).

Figure 7 – Two-time window-based detection mechanism



Source:(LU et al., 2019)

The algorithm inputs are a confidence value δ and a stream of values $x1, x2...x_t$, which can be the model accuracy measurements after each classification. Let n represents the length of W, μ_w the average of the elements in W, and μ_w the average of μ_t for $t \in W$ (BIFET; GAVALDÀ, 2007). As soon as the windows exhibit different averages, it concludes the corresponding distributions are different and the older window is dropped. To determine how the algorithm creates the two windows partitions, consider n_0 and n_1 the lengths of W_0 and W_1, and n the length of W, leading to $n = n_0 + n_1$.

$$m = \frac{1}{\frac{1}{n_0} + \frac{1}{n_1}}$$

$$\delta' = \delta / n$$

$$\epsilon_{cut} = \sqrt{\frac{1}{2m} \frac{4}{\delta'}}$$

The main limitations are memory and processing time requirements. ADWIN is unsuitable for detecting gradual concept drift since all the samples in the window have the same weight. The algorithm of this technique is described in Algorithm 1.

Algorithm 1 ADWIN: ADaptive WINdowing Algorithm

```
Initialize Window W
for each t > 0 do
    {X_t} ∪ W → W (i.e., add X_t to the head of W)
    repeat
        Drop elements from the tail of W
    until |μ̂_W0 − μ̂_W1| < ε_cut holds
        for every split of W into W = W_0 · W_1
    output μ̂_W
end for
```

ADWIN is an algorithm that represents the error rate-based (also called performance-based) strategy, which is a category that includes other recent works such as Fuzzy Windowing Drift Detection Method (FW-DDM), Dynamic Extreme Learning Machine (DELM), Wilcoxon rank sum test drift detector (WSTD), and Fisher's exact test (LIU; ZHANG; LU, 2017) (XU; WANG, 2017) (BARROS; HIDALGO; CABRAL, 2018) (CABRAL; BARROS, 2018). In addition, there are also data distribution-based algorithms that use a metric to quantify the similarity between distributions. Two of them are Local Drift Degree-based Density Synchronized Drift Adaptation (LDD-DSDA) and Equal Density Estimation (EDE) (LIU et al., 2017) (GU et al., 2016).

2.6 SUMMARY

This chapter explained the background concepts related to the research. It gives an overview of edge computing, CEP, and computational offloading areas. Moreover, other fundamental concepts to this research were explored, such as policy-based mechanism, online machine learning and concept drift. This foundation aims to make the research more understandable.

3 DRIFT ADAPTIVE OFFLOADING SYSTEM (DAOS)

This chapter describes the Drift Adaptive Offloading System (DAOS). Firstly, section 3.1 explains the high-level requirements and the architecture behind it, including the strategies for deciding *when* to offload the CEP application. Lastly, section 3.2 explains the implementation and internal mechanisms.

3.1 DESIGN GOALS AND ARCHITECTURE

Most of the offloading systems from the literature are general-purpose or built for a particular type of application, such as CEP (MURICY; JUNIOR, 2018) (FLORES et al., 2015) (FONSECA; FERRAZ; GAMA, 2018). Nevertheless, they usually share the same architectural principles and components. The idea behind DAOS is to reuse the same ideas and introduce the components that will make it intelligent and adaptive to offload CEP workloads. This section discusses the aspects related to those ideas, specifying the reasons and motivations.

3.1.1 High-Level Requirements

The architecture follows the principle of enabling DAOS to operate on the infrastructure layer of CEP applications. In this way, it can support different engines and applications, interacting with them to control the life cycle of the jobs and extract their current state. For instance, a DAOS instance monitors a fraud detection application based on CEP to decide whether the processing should be offloaded. The term system refers to the DAOS instance without considering the CEP applications.

Its decision mechanism uses online machine learning to know when to offload CEP applications between devices that are remotely connected, extending what was proposed (FONSECA; FERRAZ; GAMA, 2016). It concentrates on meeting the requirements to understand when to offload the processing to a resourceful node. Consequently, the nodes involved in the processing have to be monitored through profilers, generating contextual data to train the online learning algorithm. This algorithm gives a binary response (yes or no) about the advantage of executing the offload at a particular time. The high-level requirements are described below:

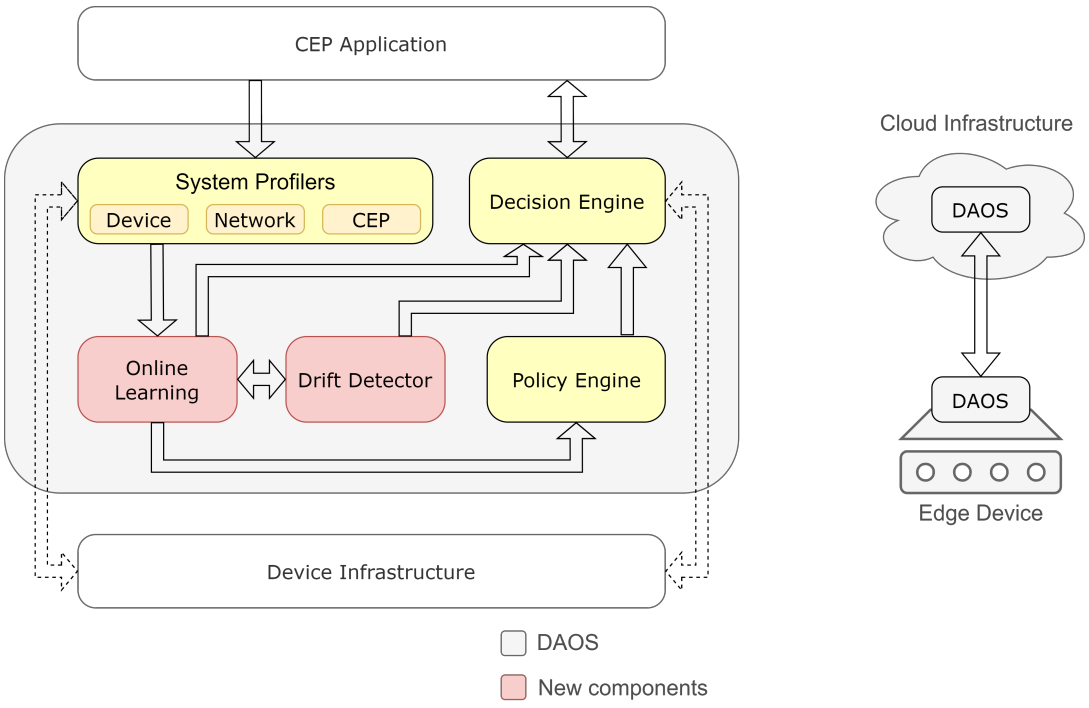
- **State Representation**: the system should be aware of local and remote node states, storing the necessary information to take the offloading decision. The state represents the overall context of the offloading system.

- **Contextual Data**: the system should have data about the resource usage of both edge and cloud environments. It also includes information about the network and the CEP application.
- **Application Control**: the system should control the execution of the CEP application, enabling it to start, stop and take snapshots from its current state. The state is transferred to the remote node before the application interruption.
- **Intelligent Decision**: the system should train a machine learning algorithm that will take a set of Contextual Data and decide if the offloading is acceptable.

3.1.2 Architecture

In traditional offloading architectures, the main components are a decision engine and supporting components such as code and system profilers. The decision engine uses the profiling data to decide which piece of code is costly and to have a notion about device and network conditions. This architecture is also adapted depending on the application requirements, in some cases leveraging the decision engine to support offloading prediction with machine learning techniques (REGO et al., 2017). Another aspect of state-of-the-art architecture is not being designed only for code offloading. It can now vary from network tasks to event processing or any *offloadable* workloads.

Figure 8 – DAOS Architecture



Source: Author

DAOS is built on top of the mentioned architectures, supporting the offloading of CEP applications in edge computing settings. Its architecture has the same components

mentioned above with an online learning algorithm that classifies the actual context information to indicate an offloading situation. Additionally, DAOS uses a concept drift detector to adapt to changes in the distribution of the underlying profiling data. It also supports the policy-based engine from (FONSECA; FERRAZ; GAMA, 2016), responsible for comparing the contextual data with resource limits. This component has the role of helping in the classifier training and acts as a fallback mechanism when the online learning models are not reliable.

The architecture is illustrated in Figure 8. It shows the interaction between the components involved in the offloading decision. Firstly, System Profilers collect metrics from the CEP Application (1) and the Device (8), sending them to the Online Learning (2) service to train the model. Secondly, the training occurs with the help of Policy Engine (3), which is responsible for categorizing the incoming profiling events. Lastly, after obtaining a good accuracy, the model sends notifications to the Decision Engine, indicating the system is demanding an offload (4). The desired accuracy is a configurable parameter of the Online Learning component, and it is compared with the model accuracy, which is updated for each instance that arrives.

Otherwise, the Decision Engine will take the notifications from the Policy Engine as a fallback mechanism for indicating an offload demand (5). This fallback mechanism is also triggered when the Drift Detector detects a change in data distribution (6). It can identify those changes and notify the interested components to make adaptations. The adaptations considered in the architecture design are switching the decision algorithm to use static policies when the models are unreliable and retraining the model to cover the new data distribution. Consequently, the decisions are based on hard limits defined by the policies. In addition, as the current training is only restarted, no additional cost is added to the system.

After receiving the events indicating an offloading situation (4) (5), the Decision Engine initiates the process internally through a state machine. It communicates with the CEP Application directly to send the incoming application events when the offloading is not happening (7). On the contrary, it makes a series of requests to the CEP applications to extract its internal NFA state and restore it in a most resourceful node. This communication to external nodes passes through the Device Infrastructure layer, which manages all networking capabilities (8).

3.1.3 Components

As shown in Figure 8, the architecture structures the components according to their relationships in the system. The Decision Engine behaves in the same way in both edge and cloud environments, receiving all the offloading messages. The difference between each environment is that the cloud uses profilers as support to accept the offload requested by the edge node, without using online learning. The components are described below:

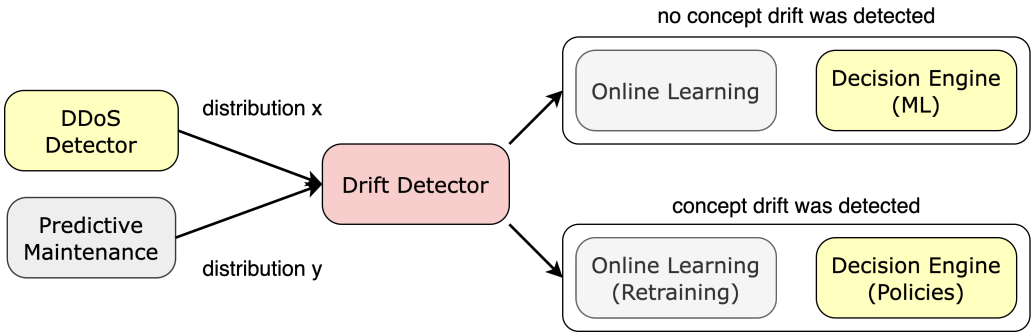
- **System Profilers**: They collect metrics from device, network, and CEP engine, which are used in the classification and training processes.
- **Online Learning**: It uses the trained model to classify the need to offload the processing to a resourceful node.
- **Drift Detector**: It receives classification assessment to identify if a concept drift occurred in any of the models. As a result, it can replace the existing models and notify the Decision Engine that the existing models are unreliable to drive the decision process.
- **Decision Engine**: It is responsible for deciding when to offload the processing to a resourceful node and managing the life cycle of CEP jobs and operators, which includes initializing or removing them, taking save points, and restoring them on the remote node. In order to help in the decision, it uses the classifier output and other contextual data to decide if the system should execute the offloading. An example of additional context information is a concept drift occurrence, which can motivate an adaptation on the decision algorithm to prioritize the static policies when deciding the need for offloading.
- **Policy Engine**: It receives the profiler metrics and compares them with resource thresholds, publishing an event if the policies are violated. For instance, it can send a violation event when the current CPU consumption is higher than 85%.

3.1.4 Fallback Mechanism

Network systems use fallback mechanisms for switching between connectivity modes according to certain situations (WU et al., 2020). Based on this mechanism, DAOS enables switching the strategies to offload CEP processing between edge and cloud nodes. Figure 9 illustrates how DAOS works when facing a scenario where two applications (DDoS Detector and Predictive Maintenance) generate two different data distributions (x and y). The difference between them can lead to a concept drift, which the Drift Detector can identify. The online learning model is retrained when drift is detected, and the decision approach is changed to work with static policies. On the contrary, the Decision Engine uses the same online learning model.

The fallback mechanism extends the adaptive approach to be secure against changes in data distribution (drifts). Besides, this primary advantage also enables the offloading system to be automatically prepared for new applications. This mechanism has the potential of being a permanent layer of general offloading infrastructure, adapting to any application that has an offloadable workload.

Figure 9 – Fallback Mechanism



Source: Author

3.2 IMPLEMENTATION

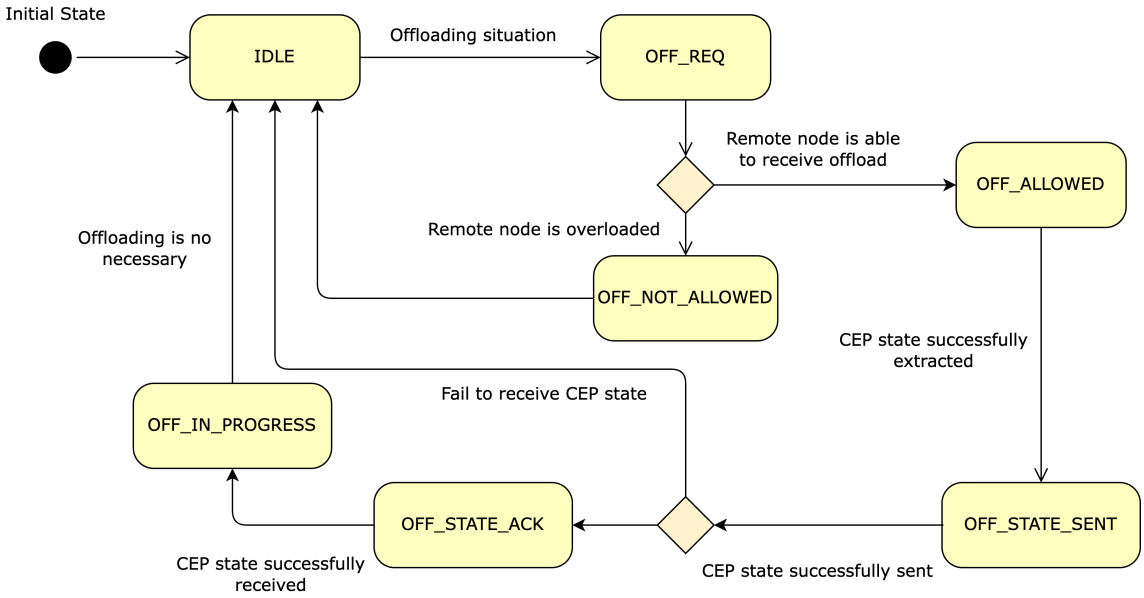
The implementation of DAOS follow the architectural requirements defined in section 3.1.1. A Proof of Concept (PoC) software validates the offloading mechanism proposed in this work. Despite being designed to support horizontal offloading, the PoC only implements vertical offloading for simplicity. This section describes the implementation and the tools/libraries used to make it possible.

3.2.1 Decision Engine

Offloading systems are designed to overcome the limitations of mobile devices by using cloud computing resources (HAGHIGHI; MOAYEDIAN, 2018). Most of these systems take into consideration the context of the device, which are usually the network and computation conditions (ZHOU et al., 2017). The implementation presented in this research uses the device context as the main factor to decide when the offload must happen. It is modeled through state machines, which are abstractions that represent one state at a given time, transiting based on external events (WANG; TEPFENHART, 2019). It can be seen in Figure 10.

The state machine has an IDLE state that indicates the engine has received no offloading events. After that, when an offloading notification arrives, the state is changed to OFF_REQ, indicating the remote should allow the offloading when viable. If allowed, it transits to OFF_ALLOWED, and the engine will extract the local CEP state and send it to the remote node. Then, after receiving the state's acknowledgment from the remote node, the state updated to OFF_IN_PROGRESS, starting the offload. In case of failure when communicating with the remote node, the state is transited back to IDLE. Despite failures in this process, there is no data loss since a local buffer stores the events until fully transiting the states. Moreover, there is no retry mechanism implemented in each state.

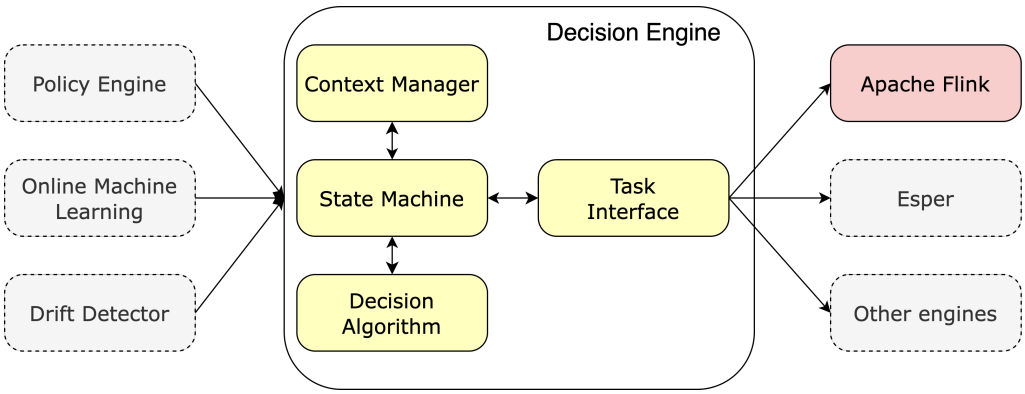
Figure 10 – State Machine Implemented by the Decision Engine



Source: Author

Under the hood, the Decision Engine has components for managing its most important capabilities: state machine, contextual data, and offloading decisions. These low-level components can be seen in Figure 11. In practice, the Decision Engine is a service that receives external events in Message Queue Telemetry Transport (MQTT) protocol and interacts with a Data Stream Processing Engine (DSPE) to orchestrate the application offloading when necessary. The interaction with CEP applications happens through the Task Interface component, which abstracts the technology used on the other side. As shown in Figure 11, this implementation uses the Apache Flink as DSPE, but it could support other engines such as Esper or Kafka.

Figure 11 – Low-Level Architecture of the Decision Engine



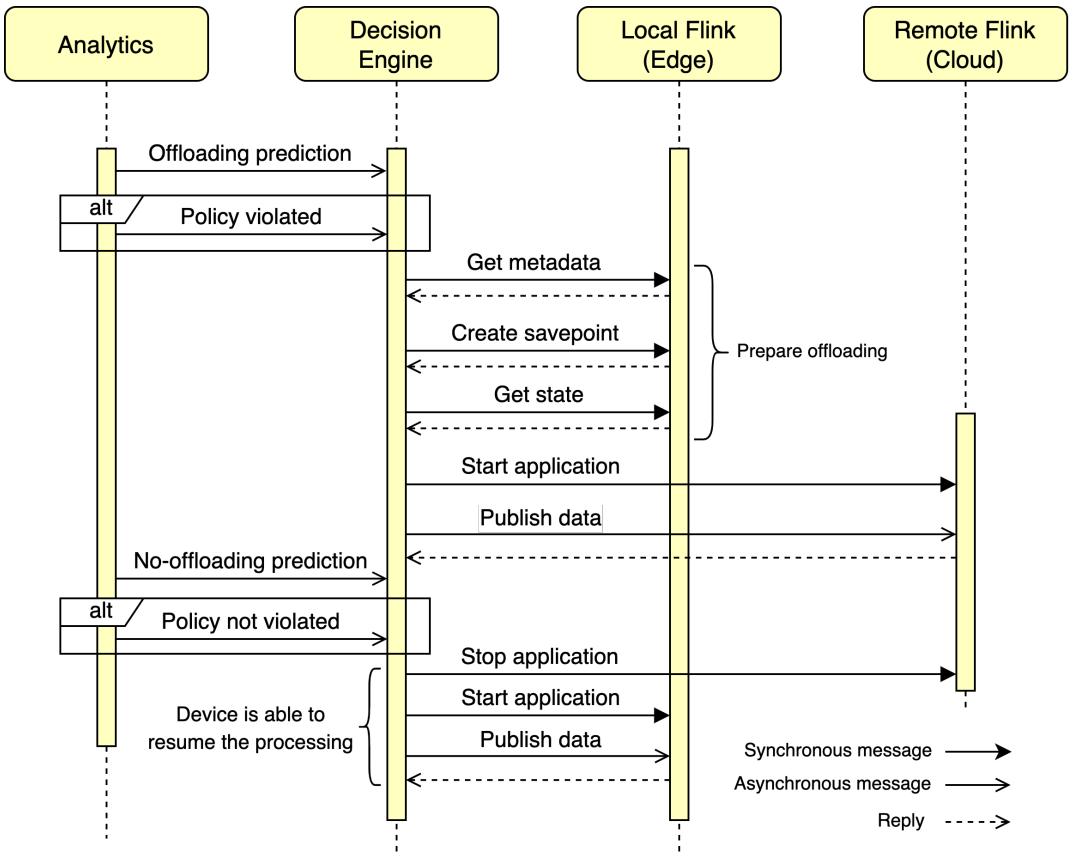
Source: Author

Figure 12 represents a sequence diagram of operations related to the offloading process.

The Analytics package represents both Online Learning and Policy Engine components. As explained in the state machine illustration, the Decision Engine receives an offloading alert and then triggers a set of requests to get the CEP application state. The cloud instance receives the edge state and starts a new application in an Apache Flink cluster. Moreover, the incoming application data is locally buffered while the cloud does not confirm the application start.

On the other hand, the Decision Engine can also receive an event indicating the offloading is no longer necessary, which means the edge device has enough resources to resume the processing. In this case, the Decision Engine starts buffering the application data locally and sends a stop message to the remote node. After receiving the stop confirmation, it restarts the application locally and redirects the data to it along with the temporary buffer.

Figure 12 – Sequence Diagram of Offloading Operations



Source: Author

3.2.2 Profiler

According to (ABOWD et al., 1999), context-aware systems use context to provide task-relevant information and services to a user, in which context is any information that describes the situation of an entity. Offloading mechanisms usually depend on contex-

tual data to make the correct decision, considering the environmental conditions such as memory consumption and network latency (MACH; BECVAR, 2017). In DAOS, the system profilers generate the context input to the Online Learning and Policy Engine components. It serves to train the online learning models to make actual predictions. Besides, the Policy Engine can compare this input to the resource thresholds to find violations.

The contextual elements are grouped into three categories: device, network, and CEP application. The first one represents the metrics obtained from the device, such as CPU and memory. Otherwise, the network category covers metrics related to the bandwidth. Lastly, the CEP contextual elements are related to the operations inside the CEP engine as the latency between two or more operators. Table 2 shows the metrics collected by the profiler. It represents a subset of the possible metrics in this environment, sufficient for making offloading decisions.

Table 2 – Contextual elements collected by the system profiler

Name	**Description**
CPU	CPU average consumption
Memory	Memory average consumption
Bandwidth	Network bandwidth between the nodes
Latency	Latency between CEP operators

Source: Author

In this work, the profiler service is a Go daemon that runs continuously in both edge and cloud. It uses a wrapper to the well-known psutil[1] library in order to obtain device metrics. Moreover, to collect information about network bandwidth, it uses a wrapper to the iperf[2] tool. Finally, to gather metrics from the Apache Flink, it was necessary to use Flink's REST API and configure the application to enable latency monitoring feature. A drawback of enabling latency monitoring is that it can decrease application performance by measuring the time a modified record takes from the source operator to the sink.

3.2.3 Online Learning

DAOS uses online learning algorithms to learn the best decision with the obtained contextual data to make an intelligent offloading decision. This learning step depends on training the ML algorithms with a dataset of instances depending on the target domain. One disadvantage of training ML algorithms with historical data is that it can become outdated if something changes on data distribution. Another aspect important in edge computing is that device's memory is usually constrained, restricting memory-intensive algorithms. As a solution, some algorithms learn incrementally and are adaptive to changes over time.

[1] https://github.com/shirou/gopsutil
[2] https://iperf.fr/iperf-download.php

Online learning techniques focus on data streaming scenarios and can use one instance at a time, avoiding previous independent training and huge memory usage. For instance, online decision tree algorithms can grow without storing the whole tree in memory. This research uses the scikit-multiflow[3], which is an open-source library that extends the well-known scikit-learn[4] framework with online learning algorithms. It supports algorithms such as decision trees, neural networks, and has modules for simulating a data stream and evaluating the classifiers in a stream fashion.

3.2.4 Drift Detector

As mentioned in the previous section, the change in data distribution can make the models outdated and compromise their performance. These changes are known as concept drifts and occur when the properties of a target variable change. In the context of CEP offloading, the target variable is the offloading prediction itself, and the properties are the contextual data collected from the environment. The concept drift can result from many situations, including changes in the CEP applications and the subsequent causes in the processing and network conditions.

DAOS makes use of the ADWIN algorithm for detecting concept drifts on the incoming profiling data, which is available in scikit-multiflow[5] library. Despite having other algorithms that perform better than ADWIN, it was selected for being widely adopted. After detecting drifts, DAOS retrains the models with the new data and changes the offloading decision strategy to policy-based until the model becomes reliable again. The fallback mechanism controls this change by checking if the model performance is acceptable. For example, it can switch the strategy back to online learning when the accuracy and recall are greater than 90%.

3.2.5 Policy Engine

The policy-based mechanism is a module that is part of the Online Learning service and acts primarily during the training and prediction phases of the algorithms. The module starts by reading the policies defined in an XML file, following the specification explained in section 2.2.1. Internally, a data structure is filled with those policies, enabling the underlying service to use them to find policy violations, which comes from matching the profiling metrics and the policies. The occurrence of policy violations triggers a function responsible for categorizing the profiling instance in Offloading (1) or No-Offloading (0). In addition, the service publishes the policy violation results to the Decision Engine component to be used by the fallback mechanism.

[3] https://scikit-multiflow.readthedocs.io/en/stable/
[4] https://scikit-learn.org/
[5] https://scikit-multiflow.readthedocs.io/en/stable/

3.2.6 CEP Engine

CEP Engine is a platform that detects patterns in an incoming event stream, comparing it against a set of rules. To support this capability, Apache Flink provides the FlinkCEP[6] library, which runs on top of the general-purpose platform. It extends the already provided Flink's DataStream API to increase the expressiveness of the CEP rules. As explained in section 3.2.1, the interaction between Decision Engine and CEP occurs through a Task API. It defines an interface with the basic operations to manage CEP jobs, such as start, stop and get state. Consequently, even with a DSPE that does not have CEP as a first-class citizen, the support can exist by extending this API.

3.3 SUMMARY

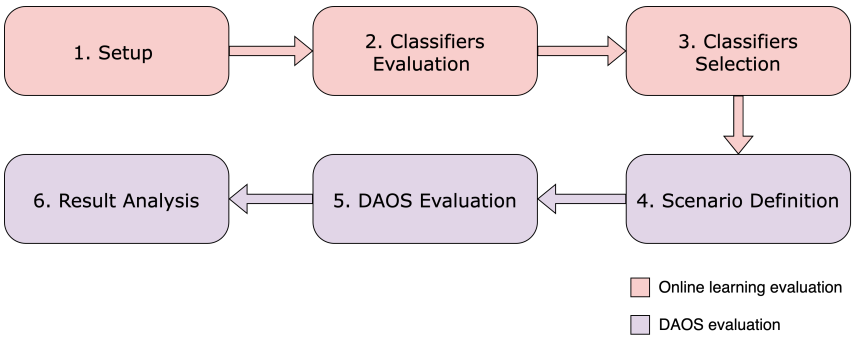
This chapter discussed the architectural proposal of DAOS, a drift adaptive offloading system that intelligently offloads CEP applications in edge computing environments (section 3.1). The architecture was designed according to the existing offloading works in the edge computing literature, reusing some of the main components. The main contribution was adding online learning, drift detection, and a fallback mechanism to protect the decisions against incorrect predictions. In addition, section 3.2 presented the internal implementation of a PoC that aims to validate DAOS through a performance evaluation.

[6] https://ci.apache.org/projects/flink/flink-docs-release-1.13/docs/libs/cep/

4 EXPERIMENT

This chapter describes the experiment designed to evaluate the offloading system, which is divided into two sections. Section 4.1 describes the performance evaluation of the online learning classifiers. Section 4.2 describes the performance evaluation of the adaptive offloading system. Figure 13 shows an overview of the experiment steps.

Figure 13 – Experiment Steps



Source: Author

The separation of both evaluations aims to facilitate the understanding of the results. It provides a clear path to select the classification algorithm used in DAOS. Moreover, the online learning evaluation has its methodologies and approaches, which cannot be mixed with a traditional performance evaluation.

4.1 EVALUATION OF ONLINE LEARNING CLASSIFIERS

This section describes the experiment that evaluates and chooses the classification model used in the proposed offloading system. It is organized according to the Table 3.

Table 3 – Sections of the online learning evaluation

Step	**Sections**
Setup	4.1.1
Results	4.1.2
Statistical Significance	4.1.3
Conclusion	4.1.4

Source: Author

4.1.1 Setup

One of the main capabilities of DAOS is the usage of an online learning classifier to help in the offloading decision process. It is incrementally trained with a stream of profiling events, giving a binary result as the output: Offloading or No-Offloading. However, the evaluation of online classifiers is complex since the data stream is unbounded, making the traditional evaluation techniques such as cross-validation unsuitable for representing this stream setting. To solve that, some techniques such as Prequential (predictive sequential) try to evaluate them in a more controlled way.

Prequential is a method to evaluate any classification algorithms in data streaming scenarios (GAMA; SEBASTIÃO; RODRIGUES, 2013). It uses each instance as it arrives in the streaming to test the model before training. Prequential guarantees the accuracy will be incrementally increased and does not require any holdout configuration, using all the available data. The main disadvantage of this approach is that it makes it hard to measure training and testing separately (GABER; ZASLAVSKY; KRISHNASWAMY, 2009). This experiment uses the default configuration of the evaluation library, which sets the Basic Window (BW) as the classical implementation for prequential evaluation. In this variation, the model is built with all processed instances of the stream, providing an average hit rate model.

The classifiers used in this evaluation are Hoeffding Tree (HT), Extremely Fast Decision Tree (EFDT), Naive Bayes (NB), and KNN (K-Nearest Neighbor), which were selected according to their different properties. Section 2.4 explains the details of each classifier.

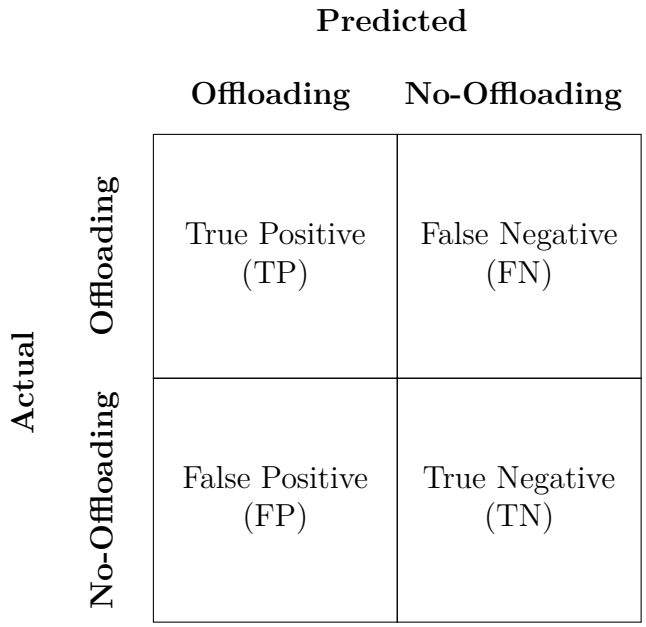
The evaluation uses metrics from the confusion matrix illustrated by Figure **??**. It compares the target values with the ones predicted by the classifier, which helps to understand the kind of errors they are making. There are two possible target values in this experiment: Offloading and No-Offloading. The rows represent the actual values, and the columns represent the predicted values. True Positive (TP) means both actual and predicted values were Offloading. True Negative (TN) means both actual and predicted values were No-Offloading. False Positive (FP) means the actual value was No-Offloading, but the classifier predicted as Offloading, also named as Type I error. False Negative (FN) means the actual value was Offloading, but the classifier predicted as No-Offloading, also named as Type II error.

According to the Confusion Matrix visualization, the following metrics can be obtained and used for evaluating the model's performance:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

Figure 14 – Confusion Matrix



Source: Author

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

$$F1 = \frac{2TP}{(2TP + FP + FN)} \tag{4.4}$$

$$Kappa = \frac{p_0 - p_e}{1 - p_e} \tag{4.5}$$

where p_0 is maximum accuracy reached by the classifier and p_e is the agreement level between the classifications. The metrics are explained below:

- **Accuracy:** ratio of correct predictions with respect to the total number of predictions.
- **Precision:** ratio of positive predictions that were done correctly.
- **Recall:** ratio of actual positive predictions that were done correctly.
- **F1:** harmonic mean of both precision and recall metrics.
- **Kappa:** how close are the predictions done with respect to the actual values.

Those metrics give a complete overview of the classifier performance, enabling us to examine the most relevant aspects of this research. In this case, reducing the Type II error when predicting Offloading and No-Offloading is desirable. The trade-off between Type I

and Type II errors relates to the cost of wrong prediction to the system as a whole. Type I error happens when the prediction is Offloading, but the actual value is No-Offloading, leading to an earlier offloading. Otherwise, Type II error occurs when the prediction is No-Offloading, but the actual value is Offloading, preventing the device from offloading before reaching the maximum capacity. Therefore, Type II error is more critical than Type I error in this scenario since the device would reach the maximum capacity.

Recall and F1-score are the metrics that most reflect the influence of Type I and Type II errors. Consequently, the evaluation and real usage of the classifiers should pay attention to both values.

Cohen's Kappa is another important metric that states the performance of a classifier compared to a random guess based on the frequency of each class (BIFET et al., 2015). It is an excellent option to assess the performance when using an imbalanced dataset, which is the case of this research because the number of No-Offloading instances is greater than the contrary.

4.1.2 Results

The evaluation was executed with 5.000 instances of the labeled dataset that contains profiling metrics from the edge devices while running a CEP application for Distributed Denial of Service (DDoS) detection. A sample of this dataset is publicly accessible in GitHub[1]. Moreover, to guarantee minimum statistical reliability, the evaluation performed 30 repetitions (KWAK; KIM, 2017). Table 4 shows the metric values for each evaluated model. The number of neighbors (k) considered in the KNN classifier was 5, the default value in the scikit-multiflow tool.

Table 4 – Results of the online learning evaluation: average values for each metric

Classifier	**Accuracy**	**Precision**	**Recall**	**F1**	**Kappa**
HT	0.9952	1.0	0.8969	0.8944	0.9456
EFDT	0.9902	0.8964	0.8924	0.8944	0.8892
NB	0.9827	0.9930	0.6323	0.7726	0.7641
KNN	0.9719	0.9151	0.4350	0.5897	0.5770

Source: Author

The results show that the HT model performs better than the others, while the results are similar to the EFDT classifier. Both algorithms share the same idea of constructing a tree incrementally and splitting when they find the best split possible. According to (MANAPRAGADA; WEBB; SALEHI, 2018), HT is more efficient computationally, while the EDFT is more efficient statistically because it allows splitting the nodes before finding the

[1] https://gist.github.com/netoax/074bce788a73fe546d4ee39abb0fac1a

best option. The EDFT usually outperforms the HT when the data follows a stationary distribution in particular scenarios.

On the other hand, NB and KNN classifiers presented the worst performance between the evaluated models. The NB classifier is usually a simple choice because it assumes independence between the attributes, which is not always true. Despite being outperformed by decision tree techniques, it works well to solve many classification problems. The accuracy of NB was satisfactory, but the other metrics show it could easily lead to Type I or Type II errors. Lastly, the KNN is an inefficient type of classifier because of its lazy characteristics, demanding storing some data in memory. Also, it depends on configuring a good value for the number of neighbors (k), which was probably the reason behind its poor performance. For the sake of simplicity, no holdout or folding evaluation was done to find the best value for 'k' in this research.

4.1.3 Statistical Significance

Table 4 gives an overview of the performance of each classifier, showing HT and EFDT as the best performing ones. However, it cannot lead to an accurate conclusion since some of the values are close. One solution to solving this type of conflict is running a statistical significance test. There are two types of tests: parametric and non-parametric. The parametric techniques assume that the data distribution is normal. As an alternative, the non-parametric techniques do not make strong assumptions about the form of the data distribution, which is an excellent fit when there is a lot of data and no prior knowledge about it. In this context, the evaluation requires a non-parametric technique that is suitable for comparing whether all the samples have the same distribution or not.

Based on this, the McNemar's test is a non-parametric technique widely used in data streaming literature for comparing two classifiers (GAMA; SEBASTIÃO; RODRIGUES, 2013) (LU et al., 2019). It was selected because the easy usage and the fact it was an acceptable type I error (DIETTERICH, 1998). This technique considers the number of instances misclassified by the first classifier (a) and not by the second (b) and vice-versa, which can be described by the following equation:

$$M = \frac{|a - b - 1|}{a + b} \tag{4.6}$$

McNemar's test follows the form of a Chi-Squared X^2 distribution and rejects the null hypothesis that classifiers have the same performance when M $>$ 3.84 with 1 degree of freedom. It was built primarily to compare two classifiers but is adaptable for more than two by using pairwise comparison. As four classifiers are being evaluated in this experiment, the test considers their combination. As shown in Table 5, a contingency table was created for each pair of models, representing the number of misclassified instances for both.

Considering a confidence level of 95% (alpha = 0.05), the hypothesis are defined below:

Table 5 – Results of the online learning evaluation: McNemar's contingency table

(a) HT x EFDT

	False	**True**
False	12	1
True	5	4782

(b) HT x NB

	False	**True**
False	10	3
True	68	4719

(c) HT x KNN

	False	**True**
False	11	2
True	212	4575

(d) EFDT x NB

	False	**True**
False	9	8
True	69	4714

(e) EFDT x KNN

	False	**True**
False	11	6
True	212	4571

(f) NB x KNN

	False	**True**
False	37	41
True	186	4536

Source: Author

- **p > alpha**: fail to reject the null hypothesis, when the models have the same proportion of errors.
- **p <= alpha**: reject the null hypothesis, when the models have a different proportion of errors.

The McNemar's statistics are described in Table 6. It indicates that the comparison between HT and EFDT classifiers fails to reject the null hypothesis, which means they have a similar proportion of errors. On the contrary, the other comparisons reject the null hypothesis, stating they do not have a similar error distribution.

Table 6 – Results of the online learning evaluation: McNemar's statistics and p-value

	Statistics	**p-value**
HT x EFDT	1.500	0.221
HT x NB	57.690	0.000
HT x KNN	204.117	0.000
EFDT x NB	46.753	0.000
EFDT x KNN	192.775	0.000
NB x KNN	91.348	0.000

Source: Author

McNemar's test has been used in this research as a convenient statistical testing approach for data streaming settings because it does not require multiple trials. However, it is known for overestimating the results in many scenarios, resulting in Type I errors (VINAGRE et al., 2021). This drawback behind McNemar's test is related to the usage of

individual instances rather than folds that represent different portions of data (BIFET et al., 2015). (GAMA; SEBASTIÃO; RODRIGUES, 2009) proposes applying McNemar's test over a sliding window to solve part of it. Otherwise, there is also a recommendation for changing the testing procedure to use folds, such as a combination of k-fold distributed bootstrap validation and Wilcoxon's signed-rank test (VINAGRE et al., 2021).

4.1.4 Conclusion

This section evaluated four online classifiers to select the most appropriate for being used in DAOS. The classifiers were initially compared in terms of performance metrics, which indicates the HT classifier has a better performance when compared to the others. In the sequence, the results were submitted to a statistical significance test called McNemar's test, which is a non-parametric test that compares the number of misclassified instances between two classifiers. As a result, McNemar's test proved that the decision tree classifiers outperform the others, indicating that both HT and EFDT could be used in DAOS. Therefore, for simplicity matter, DAOS will use the HT classifier.

4.2 PERFORMANCE EVALUATION OF OFFLOADING SYSTEM

This section presents the performance evaluation of DAOS, following the guidelines provided by (JAIN, 1990). It specifies a set of requirements to guarantee control over the experiment environment. In addition, the evaluation adopts some concepts discussed by (MORENO, 2001) on conducting Software Engineering experiments, which helps to improve the design specification. Table 7 summarize the evaluation steps.

Table 7 – Sections of the DAOS evaluation

Step	**Sections**
Goal	4.2.1
Context	4.2.2
Hypothesis	4.2.3
Variables	4.2.4
Experimental Design	4.2.6
Executions	4.2.7
Validity Threats	4.2.8
Data Analysis	4.2.9

Source: Author

4.2.1 Goal

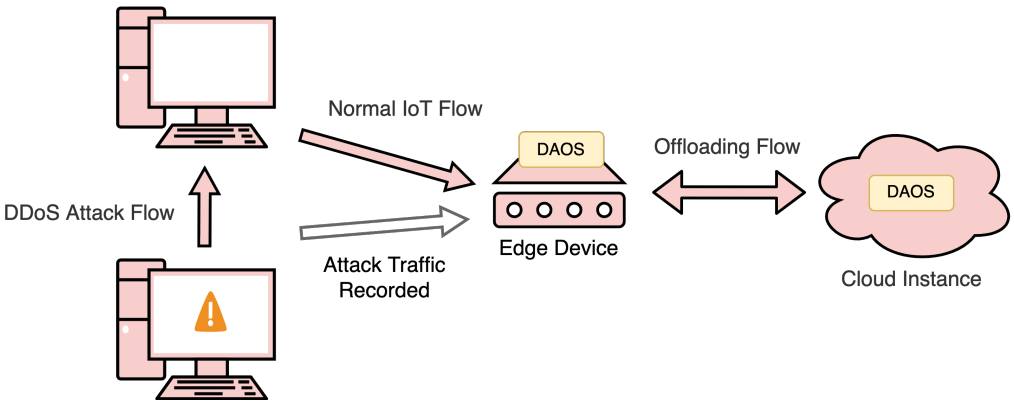
The evaluation aims to analyze the performance impact of using online learning and concept drift detection to support the decision of dynamically offloading CEP in edge computing. It assumes the best classifier for this problem was selected in section 4.1.

4.2.2 Context

The Internet of Things (IoT) is facing many challenges before being widely adopted in the market, and one of them is related to protection against Botnet attacks. This kind of attack compromises machines that are synchronized in the network and act as bots, launching a series of activities, such as Distributed Denial of Service (DDoS), Phishing, Spamming, and others (XIAO et al., 2009). In 2016, the malware known as Mirai infected many IoT devices (DVRs, webcams, and routes), causing orchestrated DDoS attacks that made platforms like Twitter and Netflix unavailable across the world (Constantinos Kolias et al., 2017).

One of the strategies for implementing an Intrusion Detection System (IDS) capable of identifying DDOS attacks is using CEP (CHEN; CHEN, 2014). The evaluation deploys DAOS in a hypothetical scenario that uses CEP for detecting DDoS attacks, potentially caused by infected machines. This application benefits from being deployed in edge devices, enabling them to collect and process network traffic data to find attacking patterns. As it is hard to collect those data, the evaluation uses the Bot-IoT dataset[2] as a reliable and realistic source of networking traffic involving IoT devices (KORONIOTIS et al., 2019). Figure 15 illustrates the scenario.

Figure 15 – Context of DAOS evaluation: experiment scenario



Source: Author

The CEP application deployed on both edge and cloud environments identifies DDoS attack patterns according to the number of TCP packets in a time window. It has two execution modes: (1) pattern rule for 128 TCP packets in 1 second (ddos-1s), and (2)

[2] https://research.unsw.edu.au/projects/bot-iot-dataset

pattern rule for 128 packets in 10 seconds (ddos-10s). These application modes generate different datasets regarding the profiling metrics, which heavily depend on the computation and network data. Consequently, concept drift can be identified in the incoming metrics as data changes according to each mode, which is appropriate for evaluating the response of DAOS to these occurrences.

With the Bot-IoT dataset on hands, the experiment simulates a flow of normal and attack packets towards the edge device. The packets are received through an MQTT broker and processed by the CEP applications, applying the rules defined in ddos-1s and ddos-10s application modes. Then, the interested clients receive a notification event when the application detects a DDoS pattern. When the offloading is necessary, the decision engine forwards this packet flow to the cloud services, and the response flow goes down in the same way. Both environments run DAOS in order to coordinate the offloading workflow.

The workload data is published in a unique stream of events to the experiments' applications. A test suite was developed to create the event stream and control each treatment's event rate. The applications receive the events and process against the CEP rules, categorized as simple and complex. The difference between them is the time window size and the number of similar packets in that window for being considered a DDoS attack.

4.2.3 Hypothesis

The statistical hypotheses are constructed to make a statistical decision about the experimental population (MORENO, 2001). Supposing two alternative solutions are being compared, it would be possible to formulate a hypothesis that there is no statistical difference between them, called the null hypothesis (denoted by H_0). The contrary leads to an alternative hypothesis that assumes the H_0 is null, often analyzing other aspects of the comparison. The following hypotheses are defined for this experiment:

H_0: There is no significant difference in performance when using DAOS compared to the policy-based mechanism.

- $H_0 : CPU_{daos} = CPU_{policy}$
- $H_0 : Memory_{daos} = Memory_{policy}$
- $H_0 : Bandwidth_{daos} = Bandwidth_{policy}$
- $H_0 : Latency_{daos} = Latency_{policy}$

H_1: The performance when using DAOS is higher compared to using the policy-based mechanism.

- $H_1 : CPU_{daos} < CPU_{policy}$

- $H_1 : Memory_{daos} < Memory_{policy}$
- $H_1 : Bandwidth_{daos} < Bandwidth_{policy}$
- $H_1 : Latency_{daos} < Latency_{policy}$

4.2.4 Variables

Response variables describe the effect of different factor levels on the experimental units (MORENO, 2001). The Goal Question Metric (GQM) is an approach that facilitates the identification of those variables in a software engineering experiment (BASILI; CALDIERA, 1994). This technique requires the definition of specific goals and questions related to them. Consequently, the question is analyzed to define which metrics answer them. The GQM questions are defined below:

G1. Resource usage optimization

- Q.1.1. How many times have the devices reached maximum capacity?
- Q.1.2. What is the performance impact of adding DAOS to the edge device?
- Q.1.3. Is the overall performance negatively affected by the online learning model?

The GQM process evidenced the high-level goal of this research, creating a relationship with the target response variable and questions that lead to quantifiable answers. **G1** involves the analysis of the efficiency when running DAOS for offloading CEP applications in the edge. As a result, it defines the following response variables:

- Efficiency in offloading CEP applications in the edge without overloading the devices (**G1**).
- Effectiveness in dynamically adapting the decision mechanism based on concept drift occurrences (**G1**).

4.2.4.1 Metrics

As a result of the GQM process, Table 8 describes the metrics responsible for answering the response variables.

The implementation of DAOS was instrumented accordingly to collect the essential metrics through the profiling service, as explained in section 3.2.2. In addition, other general variables such as interruptions and the number of offloading would be collected directly from the experiment logs. The service logs are stored in a text file for further processing and analysis.

Table 8 – Metrics of the DAOS evaluation

Metric	Description
Memory usage (%)	Average of memory the device is using
CPU usage (%)	Average of CPU the device is using
Bandwidth (MB/s)	Rate of data transfer for a fixed period of time
CEP latency (ms)	Latency between CEP operators

Source: Author

4.2.4.2 Factors and Levels

According to (MORENO, 2001), the factors are any aspects intentionally varied in the experimentation that can affect the response variables. They are associated with a set of possible alternatives or levels. For instance, supposing the evaluation that compares networking monitoring tools, the tool would be a factor, and its possible alternatives: Zabbix, Prometheus, and others.

Table 9 – Factors and alternatives of DAOS evaluation

Factors	Levels
Strategy	Policy-based, ML-enhanced, Drift-enhanced (fallback)
Throughput (events/s)	250, 500, 750
Application mode	ddos-1s, ddos-10s

Source: Author

Table 9 shows the factors and alternatives used in this experiment. The strategy is a factor that represents the mechanism under evaluation, which has three alternatives: policy-based, ML-enhanced, and Drift-enhanced. Also, the throughput is the number of events per second sent to the mechanisms. Lastly, the application mode describes the complexity of the DDoS application used in the experiment, which uses different time windows to increase the processing load.

4.2.4.3 Parameters

The parameters are properties that do not change during the experimentation (MORENO, 2001). As shown in Table 10, the parameters are mostly related to the software and hardware capabilities required for running DAOS.

Table 10 – Parameters of the DAOS evaluation

Type	Parameter
CEP Engine	Apache Flink (v1.10.0)
Hardware (Edge)	Raspberry Pi 3 Model B (Quadcore, 1 GB memory)
Operating System (Edge)	Raspberry Pi OS
Hardware (Cloud)	AWS EC2 - a1.xlarge (4 vCPU, 8 GB memory)
Operating System (Cloud)	Linux Ubuntu 18.04

Source: Author

4.2.5 Workload Characterization

According to (JAIN, 1990), there are two types of test workloads: real and synthetic. The real workloads represent the system in its everyday operations. Otherwise, the synthetic ones are suitable for evaluations, demanding more control over the workloads. Usually, the experiments use synthetic workloads due to the ability to repeat and modify them based on their needs.

As explained in the section 4.2.2, this experiment uses a workload from a hypothetical scenario of identifying DDoS attacks against IoT devices. The real effectiveness of this solution depends on other factors, such as the level of hardware security. Therefore, this experiment is concerned with the overhead aspect of the system when receiving the malicious packets at different rates. The computation offloading strategies are studied to guarantee that this detection process works effectively in an acceptable time.

Table 11 – Description of the dataset features used in DAOS evaluation

Column	Description
Stime	Record start time
Proto	Transaction protocol present in network flow
Saddr	Source IP address
Sport	Source port number
Daddr	Destination IP address
Dport	Destination port number
Bytes	Total number of bytes in transaction
State	Transaction state

Source: Author

The experiment uses the Bot-IoT dataset, which is a synthetic workload with network packets that represent both DDoS attacks and normal IoT flow (KORONIOTIS et al., 2019). It has 72.000.000 records and throughput of 42.7 events per second. The event publishing rate is controlled for exercising the applications with different load levels. Table 11

describes the dataset columns that are useful for building a CEP application that detects anomalous network traffic.

4.2.6 Design

This experiment does not have blocking variables that can affect the response variables. It can be performed by combining the alternatives of each factor described in Table 9. The only exception to this combination is when evaluating the Drift-enhanced strategy. It requires analyzing DAOS under the opposite load levels and different application modes: (ddos-1s, 250) x (ddos-10s, 750). This adaption makes it feasible to detect concept drifts in a controlled manner. According to (MORENO, 2001), this type of experiment is called Fractional Factorial Design.

4.2.7 Executions

The experiment execution follows the fractional factorial combination explained in section 4.2.6. It evaluates and compares different strategies for CEP offloading in the edge, using the variation of throughput, strategy, and application modes. In addition, to facilitate the understanding of the evaluation, the executions will be separated into three parts: policy-based, ML-enhanced, and Drift-enhanced. Finally, each execution takes 30 minutes, except those that evaluate the Drift-enhanced mechanism, which takes 120 minutes to guarantee the detection of drifts.

4.2.8 Validity Threats

According to (WOHLIN et al., 2012), software engineering experiments should be concerned with the validity of their results. In this research, the following threats were identified:

- **Construct validity:** the experimental design was adapted for reproducing a scenario with concept drift occurrences. Instead of comparing all the factors and levels with the Drift-enhanced approach, it focuses on the combination that consumes fewer resources (ddos-1s, 250) and the combination that consumes more resources (ddos-10s, 750). Hence, the causal relationship between the other combinations and the improvement with the Drift-enhanced mechanism cannot be assured.
- **Internal validity:** the experimental instrumentation uses the profiling component of DAOS to collect the expected metrics. However, it is considered a threat because profiling is one of the components under evaluation and can negatively affect the measured performance. One aspect that minimizes this problem is that DAOS uses widely adopted monitoring tools such as psutil[3] and iPerf3[4], which are wrappers to C code that executes the Unix system calls to get the metrics.

[3] https://github.com/shirou/gopsutil
[4] https://iperf.fr/

- **External validity:** the throughput is an experimental factor that represents the load directed to the edge devices. In this evaluation, the alternatives are 250, 500, 750 events per second, comprehended as low, medium, and high rates. However, these values were empirically defined based on Apache Flink's behaviors without any previous evaluation. Thus, it can influence the generalization of DAOS for other scenarios in the edge that would receive a higher rate.

4.2.9 Data Analysis

This section analyses the results of each execution. It presents the statistical data and charts with resource consumption over time, giving an overview of the data. It drives the discussion to compare the performance between intelligent approaches (ML-enhanced, Drift-enhanced) and policy-based. The executions are detailed in sections 4.2.9.1, 4.2.9.2 and 4.2.9.3.

4.2.9.1 Policy-based Offloading Mechanism

The policy-based offloading mechanism use thresholds to control when the offloading is acceptable (e.g. 80% of CPU and 60% of memory). In the face of a violation, the decision engine starts offloading the CEP application to the cloud. Based on the experimental factors in Table 9, the evaluation executes the policy-based mechanism by varying the throughput and application modes. It does not consider any machine learning solution. The executions take 30 minutes individually, being appropriate for noticing the increase in resource consumption and the subsequent offloading. Table 12 provides an overview about the collected statistical measures. In addition, Figure 16 shows the same results in a box plot format. The offloading occurrences mostly happen when the system receives a higher throughput (500, 750).

Table 12 – Results of DAOS evaluation: mean values for the policy-based mechanism in the edge

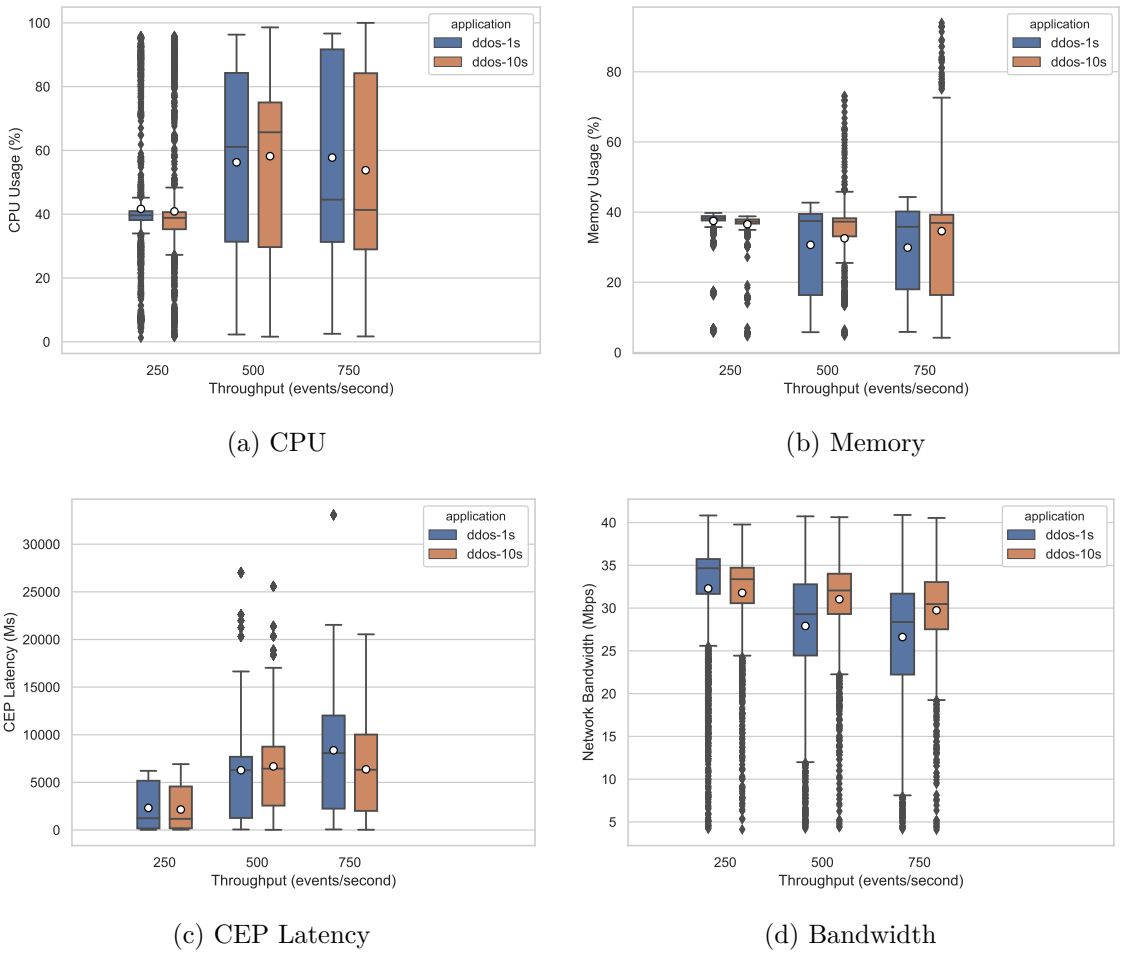
	$T = 250$		$T = 500$		$T = 750$	
	ddos-1s	ddos-10s	ddos-1s	ddos-10s	ddos-1s	ddos-10s
CPU (%)	41.67	40.88	56.27	58.19	57.76	53.79
Memory (%)	37.49	36.55	30.67	32.5	29.89	34.59
Bandwidth (MB/s)	32.29	31.77	27.91	31.03	26.61	29.74
Latency (ms)	2318.71	2150.84	6277.98	6679.54	8366.88	6370.72

Source: Author

Figure 16a shows the CPU results for the edge device. As expected, there is a considerable difference in the values when the application receives a higher throughput of events. Due to the increase of data in a smaller time window (1 second), the CEP application naturally demands more CPU to complete the processing. It becomes clear when

comparing the results for both applications, where the consumption is greater with the application that expects 128 events in 1 second. Moreover, it shows that the CPU has reached a maximum value of almost 100% in all treatments, making sense with a constrained device. No crashes or interruptions were detected despite reaching this value, probably because the offloading happens in the sequence.

Figure 16 – Results of DAOS evaluation: boxplot for policy-based mechanism (edge)



(a) CPU

(b) Memory

(c) CEP Latency

(d) Bandwidth

Source: Author

Another important metric in CEP application is the memory consumption, described in Figure 16b. It shows that average memory consumption remains quite the same even with variations in throughput and application modes. Nonetheless, the maximum consumption is reached when running the ddos-10s application with higher throughput (750). It groups 128 events in a time window of 10 seconds, demanding more memory when compared to the ddos-1s. In the same context, the CEP Latency is a metric that reflects the internal overhead of the CEP operators. Figure 16c shows that it behaves similarly to the other metrics when the throughput increases.

The networking aspect of the evaluation considers the bandwidth between the edge and cloud. Figure 16d shows that bandwidth decreases when the throughput is high.

What happens is that when the offloading starts, the decision engine redirects the data flow to the cloud instance. Consequently, the higher the throughput, the more bandwidth will be used to support that redirection.

Table 13 – Results of DAOS evaluation: mean values for the policy-based mechanism in the cloud

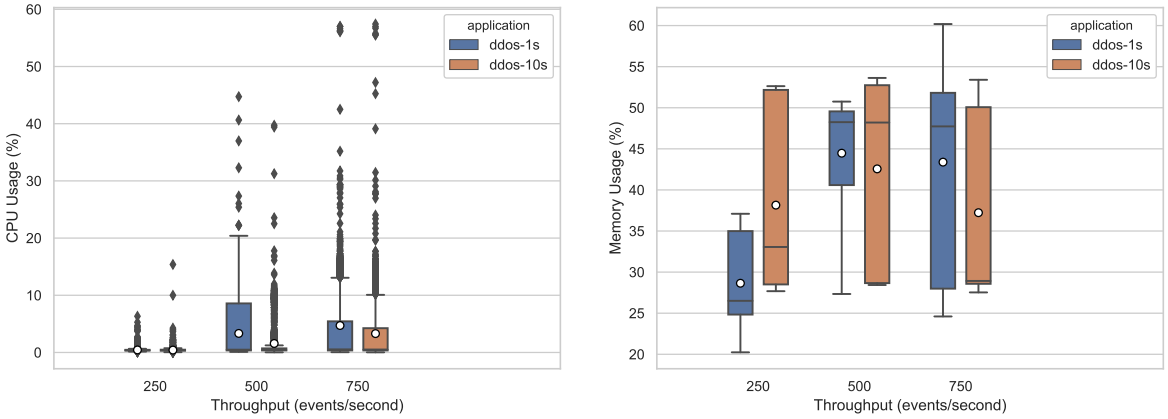
	$T = 250$		$T = 500$		$T = 750$	
	ddos-1s	ddos-10s	ddos-1s	ddos-10s	ddos-1s	ddos-10s
CPU (%)	0.44	0.42	3.32	1.57	4.71	3.29
Memory (%)	28.64	38.15	44.47	42.56	43.38	37.22

Source: Author

With respect to the cloud, Table 15 presents an overview of the statistic results, which are also shown as box plots in Figure 17. This environment is very different from the edge, considering the available resources. For that reason, the results show that the average CPU consumption remains low independently of the throughput variation. The skewed data in the boxplots are probably related to the bootstrap of the Apache Flink cluster and the other services required by DAOS.

The memory consumption results in the cloud environment are consistent with each application's requirement. It starts considerably high for the application ddos-10s and remains at the same level until the end. Otherwise, the ddos-1s application is only impacted when it receives a higher throughput.

Figure 17 – Results of DAOS evaluation: boxplot for policy-based mechanism (cloud)



(a) CPU

(b) Memory

Source: Author

4.2.9.2 ML-enhanced Offloading Mechanism

Unlike the policy-based mechanisms, the ML-enhanced approach can dynamically adapt to changes in terms of resources and applications. It uses online machine learning for learning when the offloading should happen, informing the decision engine accordingly. To implement this approach, DAOS uses the Hoeffding Tree algorithm as the online learning classifier, selected in section 4.1.

Table 14 – Results of DAOS evaluation: mean values for the ML-enhanced mechanism in the edge

	$T = 250$		$T = 500$		$T = 750$	
	ddos-1s	ddos-10s	ddos-1s	ddos-10s	ddos-1s	ddos-10s
CPU (%)	33.39	31.50	43.97	42.50	52.15	52.27
Memory (%)	23.80	31.02	24.28	28.8	25.91	29.45
Bandwidth (MB/s)	35.19	33.44	34.04	32.73	33.40	31.56
Latency (ms)	2524.60	2125.90	7224.47	6484.69	5412.35	5593.62

Source: Author

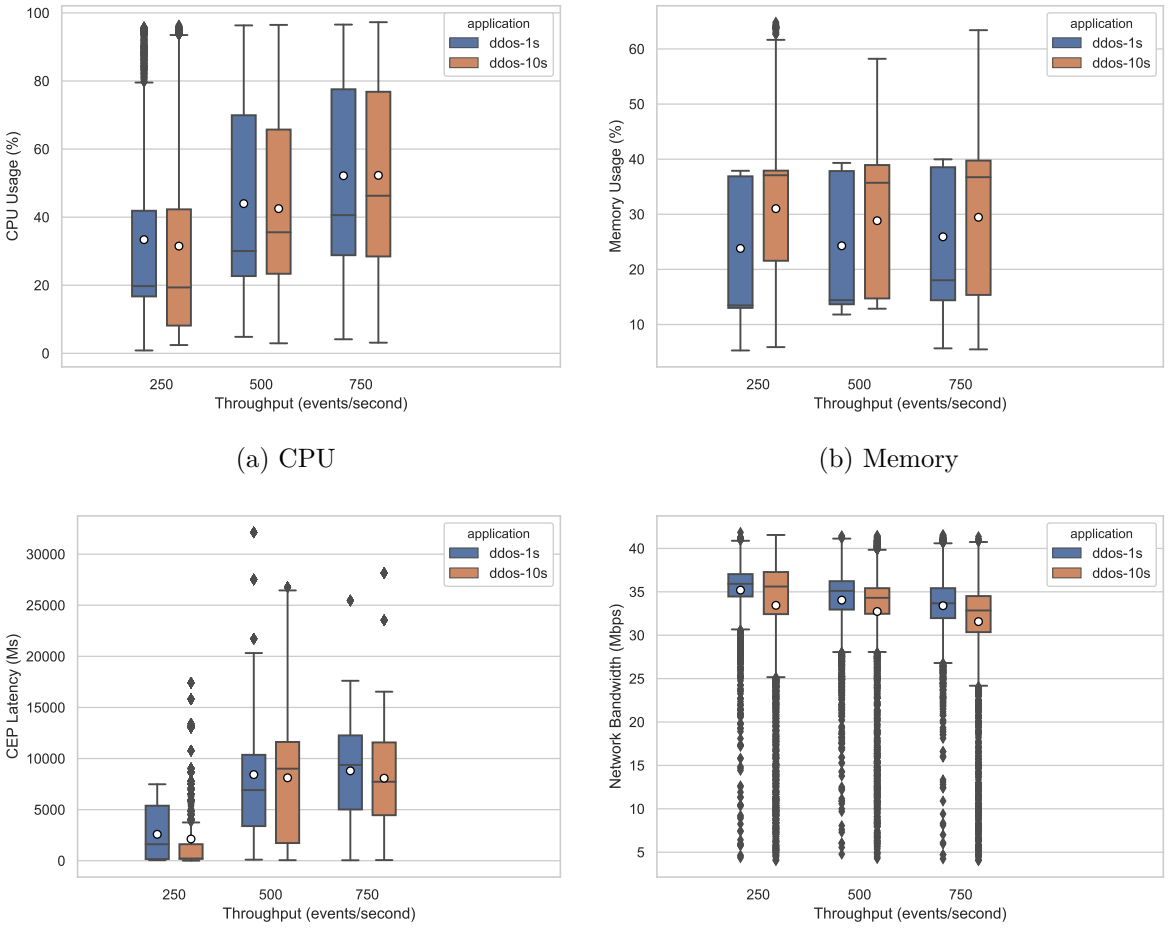
The online learning model was initially warmed-up with ten executions to reach a minimum desired performance and stabilize the system (VOJTˇ et al.,). The cost of not running offloading is high compared to the contrary, requiring an acceptable classification performance. This execution demanded a configuration on the Decision Engine to fall back the strategy to policy-based while the model is not ready yet. Based on the results shown in Table 4, it considers the performance values of 95% for accuracy and 80% for recall.

Table 14 gives an overview about the statistical measures for this ML-enhanced approach. Moreover, Figure 18 presents the same results box plot charts.

Figure 18a shows the results of CPU consumption for this mechanism on edge. The mean values are lower when compared to the policy-based mechanism, which is probably a consequence of executing an offload earlier due to classifier prediction. It happens because the policies are defined according to hard thresholds and some repetition rules, while the online learning approach has an improved boundary definition to decide in favor of offloading. Furthermore, the maximum CPU consumption has not reached 100% in any of the scenarios.

The ML-enhanced mechanism requires a new service for training and classifying with the incoming profiling events. However, Figure 18b shows it does not negatively impact memory consumption. On the contrary, the usage of online learning makes the whole system more efficient, reducing memory compared to the policy-based mechanism results. In addition, those results make clear that the Hoeffding Tree algorithm used in the system is computationally efficient, avoiding storing a huge decision tree on memory. The same

Figure 18 – Results of DAOS evaluation: boxplot for ML-enhanced mechanism (edge)



(a) CPU

(b) Memory

(c) CEP Latency

(d) Bandwidth

Source: Author

line of thought could be applied to analyze the latency results. Figure 18c indicates the average latency is lower than with the policy approach.

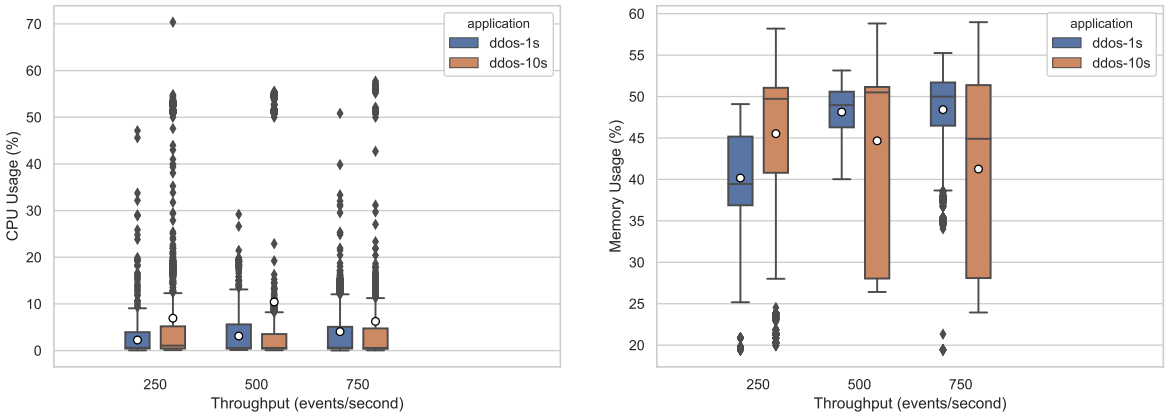
Table 15 – Results of DAOS evaluation: mean values for the ML-enhanced mechanism in the cloud

	$T = 250$		$T = 500$		$T = 750$	
	ddos-1s	ddos-10s	ddos-1s	ddos-10s	ddos-1s	ddos-10s
CPU (%)	2.25	6.93	3.12	10.41	4.06	6.25
Memory (%)	40.17	45.52	48.13	44.65	48.42	41.25

Source: Author

Figure 18d exhibits the results for bandwidth consumption between edge and cloud. Like the other metrics, bandwidth consumption has also improved with the online learning approach. That is probably related to the improvement in the system's overall performance since the network stack of each device no more excessively buffers data. In contrast to the

Figure 19 – Results of DAOS evaluation: boxplot for ML-enhanced mechanism (cloud)



(a) CPU (Cloud)

(b) Memory (Cloud)

Source: Author

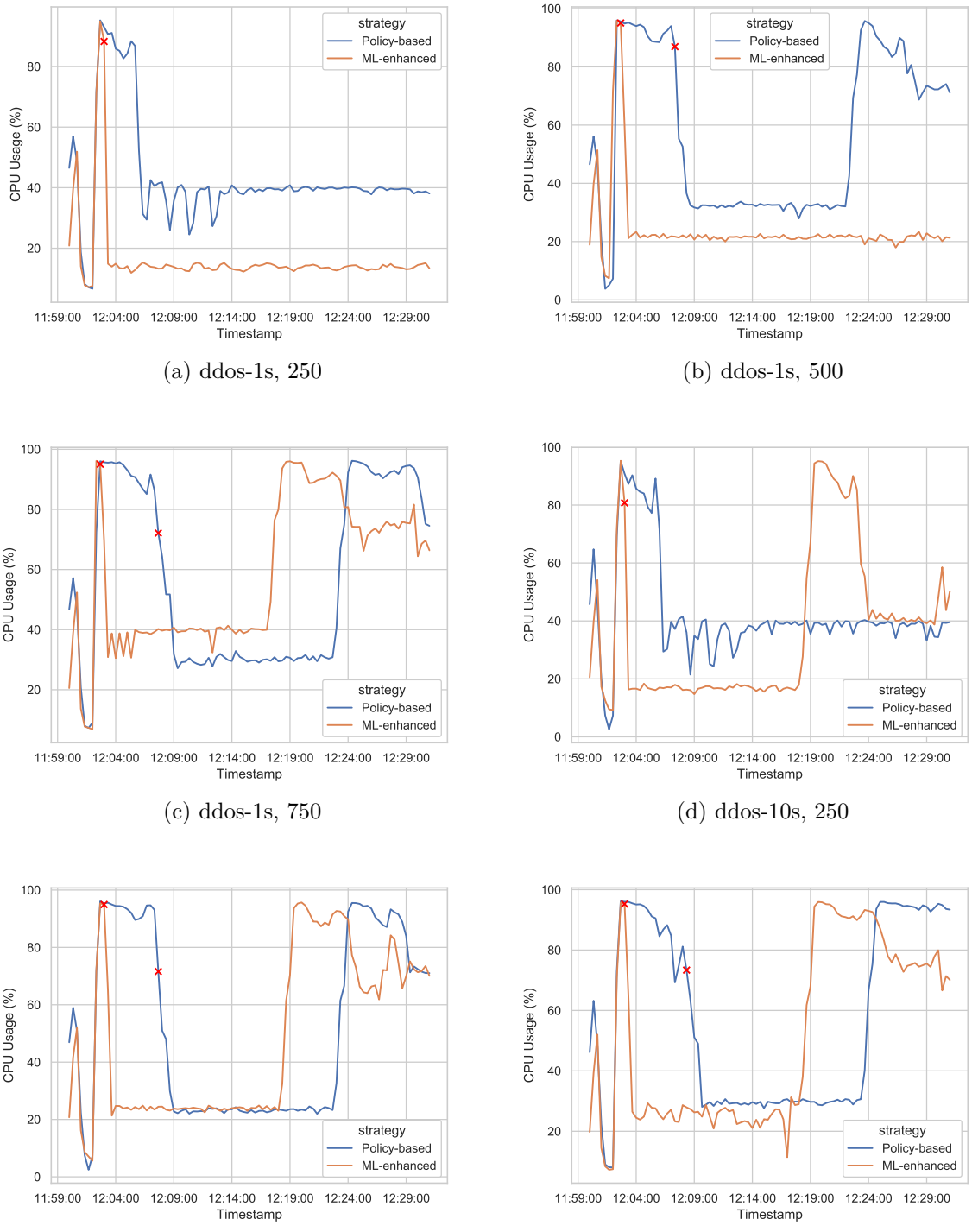
policy-based approach, the average bandwidth does not show any special variation when increased throughput.

Table 15 provides an overview of the cloud statistic measures. Also, Figure 17 shows the chart results for both CPU and memory usage on the cloud instance. Compared to the edge results, the online learning mechanism increases the load on the cloud side. As the offloading decision becomes more precise on the edge device, it reduces the idle time of the cloud instances, which is a behavior that makes sense when looking at the results.

To provide a better understanding of the offloading occurrences, Figure 20 shows the results for each pair of application modes and throughput rates concerning the edge environment. It represents only one sample from the amount of data collected. As shown in the charts, the main difference between the policy-based (blue line) and the ML-enhanced (orange line) is that the ML-enhanced strategy anticipates the offloading decision. For instance, 20b shows that the ML-enhanced strategy started an offload at the timestamp 12:02:00, while the policy-based strategy took a few minutes to make the same decision. A consequence of this anticipation is that the edge device consumes fewer resources than using a policy-based decision. Naturally, the same explanation could be extended to the other metrics monitored in the evaluation.

The improvements brought by the ML-enhanced strategy are due to the intelligence in identifying offloading situations. It presents an outstanding boundary definition reached with the trained model that uses the profiling data collected from this offloading process. The policy-based strategy has to certify that a set of metrics represent an actual offloading situation since it could be from a sporadic consumption peak. As explained in section 2.2.1, it usually uses repetition and composing rules to ensure the offloading occurrence. The policy-based strategy has methods to reach the offloading anticipation achieved by the ML-enhanced strategy. However, the main advantage of using the ML-enhanced strategy

Figure 20 – CPU usage over the time of policy-based and ML-enhanced executions. The 'x' marker indicates an offloading occurrence.



(a) ddos-1s, 250

(b) ddos-1s, 500

(c) ddos-1s, 750

(d) ddos-10s, 250

(e) ddos-10s, 500

(f) ddos-10s, 750

Source: Author

is reducing those manual optimizations, learning from the data generated by the devices.

4.2.9.3 Drift-enhanced Offloading Mechanism

The online learning model can become outdated with the deployment of a new CEP application in the edge or any change in the environment that influences profiling data distribution. Consequently, it would not be feasible to use the ML-enhanced approach until the new model reaches a minimum performance to guarantee that the offloading decisions will make sense. For that reason, the third strategy analyzed in this evaluation considers the addition of concept drift detection to the online learning workflow, using the drift occurrences as an alarm to retrain the model and fallback the decision approach to the policy-based mechanism.

The Drift-enhanced mechanism demands starting a Drift Detector component on the online learning service. Besides, it requires an experimental design adaptation to detect concept drifts in a controlled time window. The design of other approaches considers executing them in a window of 30 minutes. However, this time is not enough for detecting concept drifts in a repeated manner. Consequently, it was necessary to increase it to 120 minutes and run the opposite applications in terms of throughput and complexity to guarantee a change in data distribution.

Table 16 – Results of DAOS evaluation: mean values for the Drift-enhanced mechanism in the edge

	ddos-1s (T=250)	ddos-10s (T=750)
CPU (%)	42.79	65.52
Memory (%)	38.25	36.07
Bandwidth (MB/s)	31.20	29.14
Latency (ms)	3044.89	7717.59

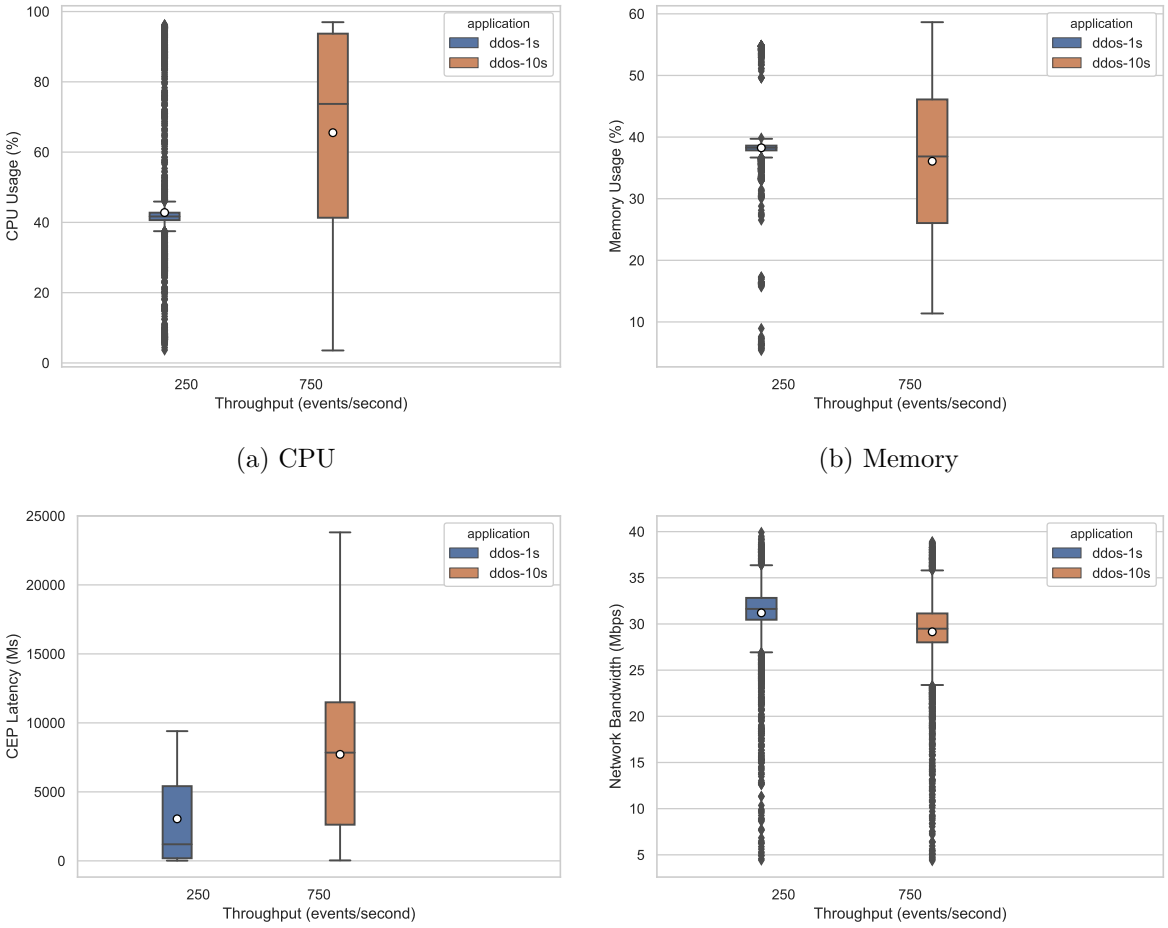
Source: Author

In the same way, Table 16 shows an overview for the edge environment, and Figure 21 shows the plotted charts. In both applications, the average CPU and memory consumption is higher when compared to the ML-enhanced mechanism, which is probably a consequence of the difference between the execution time and the additional component introduced. It is important noticing this difference was intentionally introduced through the adaption in complexity and loading characteristics to make it easier to detect concept drifts.

Despite maintaining the same throughput rates as the other mechanisms, the variation influences the CEP operator's latency. The engine may have to duplicate the garbage collector to clear the unused state. Moreover, receiving a larger number of events compromises the long-term performance. For that reason, the average latency presented in Figure 21d is greater when compared to the other mechanisms.

Adding a new component to the systems does not influence the application processing

Figure 21 – Results of DAOS evaluation: boxplot for Drift-enhanced mechanism (edge)



(a) CPU

(b) Memory

(c) CEP Latency

(d) Bandwidth

Source: Author

requirements. As a result, the bandwidth statistics presented in Table 21d are pretty similar to the results for the ML-enhanced mechanism. The small variation is probably a consequence of the opposite throughput values since the ddos-10s application offloads more data than the ddos-1s because of its internal state.

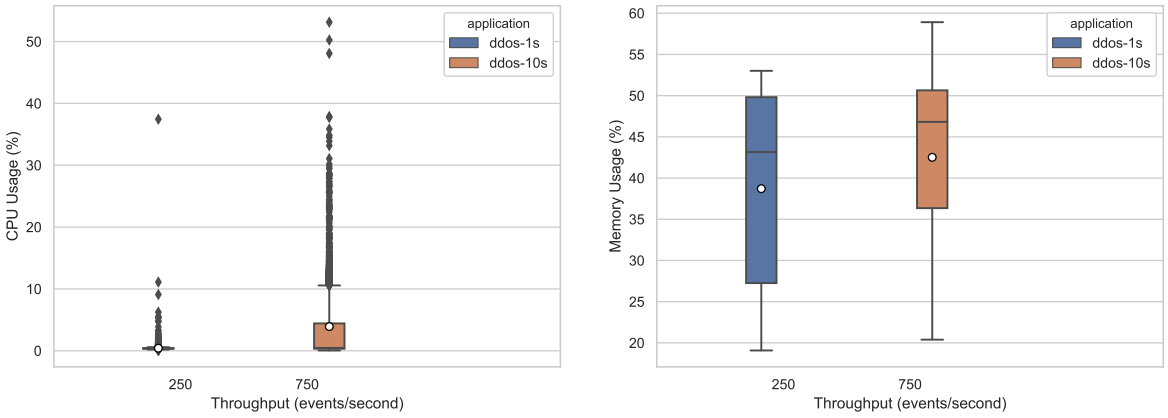
Tables 17 and Figure 22 shows the CPU and memory statistics for the cloud environment. The throughput variation introduces a natural difference between ddos-1s and ddos-10s measures that receive fewer events per second, consuming less CPU and memory.

Table 17 – Results of DAOS evaluation: mean values for the Drift-enhanced mechanism in the cloud

	ddos-1s (T=250)	ddos-10s (T=750)
CPU (%)	0.41	3.91
Memory (%)	38.71	42.51

Source: Author

Figure 22 – Results of DAOS evaluation: boxplot for Drift-enhanced mechanism (cloud)

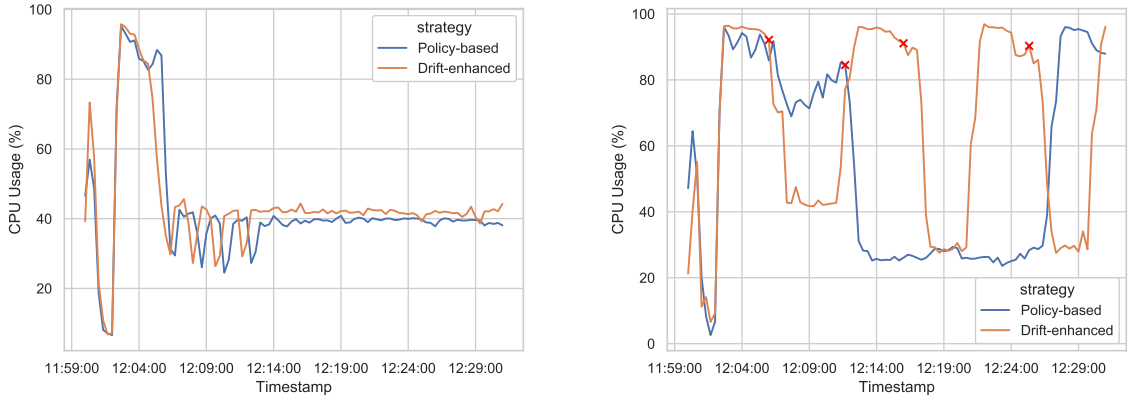


(a) CPU (b) Memory

Source: Author

The main distinction between the Drift-enhanced and the ML-enhanced mechanisms is related to the ability to adapt to concept drift occurrences, changing the offloading decision mechanism on the fly. One of the challenges when evaluating it is detecting concept drifts in a controlled manner. It can be solved by creating a considerable difference between the profiling measurements, which happens when there is a change in two factors: application mode and throughput. That said, Figure 23 evidences that difference when comparing the (ddos-1s, 250) and (ddos-10s, 750). In Figure 23b, there is a concept drift occurrence at the timestamp 12:24:37 and an offloading in the sequence. Despite facing an online model restarting, the decision fallback to the policy-based strategy and can continue with the offloading decisions.

Figure 23 – CPU usage over the time of policy-based and Drift-enhanced executions. The 'x' marker indicates an offloading occurrence.



(a) ddos-1s, 250 (b) ddos-10s, 750

Source: Author

As described in the statistical measure tables, the performance of the Drift-enhanced mechanism is this case is worse than ML-enhanced since the number of offloading is intentionally increased to provoke drifts. The offloading frequency keeps lower in the executions that evaluate policy-based and ML-enhanced mechanisms. The main reason for that is because a timeout protection mechanism avoids excessive offloading when the device node becomes suitable for processing in a short time since the last offloading. Figure 23b shows the consequence of minimizing the timeout protection as mentioned above. Even with this adjustment, the simultaneous offloading occurrences in the comparison show the Drift-enhanced also reacts a little faster when facing an offloading situation.

4.2.10 Statistical Significance

The statistical test aims to validate whether there is a statistical difference between the results presented in section 4.2.9. It is responsible for rejecting or not the hypotheses defined in section 4.2.3. Initially, a Shapiro-Wilk test was executed to state whether the results are normally distributed (SHAPIRO; WILK, 1965). This test guides the choice of the testing approach. In this test, the null hypothesis means the sample follows a normal distribution, rejected when the p-value is greater than 0.05 with a confidence level of 95%. The obtained results, in general, have a p-value lower than 0.05, rejecting the null hypothesis, which leads to a non-parametric testing approach.

Mann-Whitney U is a non-parametric test that seems appropriate for this scenario. It combines the samples and ranks them together to check the formation of opposite values clusters or to check if the values are randomly mixed (MANN; WHITNEY, 1947). The null hypothesis states there is no significant difference between the distributions. Its rejection means the samples are significantly different from each other, which occurs when the p-value is lower than 0.05 (alpha). It compares the policy-based mechanism against its two alternatives: ML-enhanced and Drift-enhanced.

- **p > alpha**: fail to reject the null hypothesis, when there is no significant difference among the samples.
- **p <= alpha**: reject the null hypothesis, when there is a significant difference among the samples.

Table 18 compares both policy-based and ML-enhanced strategies. As it can be seen, the p-values are mostly lower than 0.05, rejecting the null hypothesis. This result states that the difference between both strategies is statistically significant. Therefore, as the performance results for the compared alternatives show that the ML-enhanced strategy is more performative than its alternative, the null hypothesis rejection means it is statistically better than the policy-based strategy.

Table 18 – Mann-Whitney U test results for edge device metrics. It compares two offloading strategies: policy-based and ML-enhanced.

Application	Metric	Throughput	Statistics	p-value
ddos-1s	CPU	250	6283034.500	0.000
		500	5891694.500	0.000
		750	5536633.500	0.000
	Memory	250	8989790.000	0.000
		500	6662005.000	0.000
		750	6447125.500	0.000
	Bandwidth	250	2981075.000	0.000
		500	1672370.500	0.000
		750	1631400.500	0.000
	Latency	250	1517702.500	0.001
		500	721396.000	0.000
		750	756136.500	0.013
ddos-10s	CPU	250	12832236.000	0.000
		500	7905111.000	0.000
		750	5187863.500	0.234
	Memory	250	12008029.500	0.000
		500	6285536.500	0.017
		750	5481869.000	0.000
	Bandwidth	250	4829091.500	0.000
		500	3271659.000	0.000
		750	2941375.500	0.000
	Latency	250	5324416.500	0.000
		500	1009748.500	0.000
		750	684672.500	0.000

Source: Author

On the other hand, Table 19 compares the policy-based with the Drift-enhanced approach. As specified in section 4.2.9.3, this comparison is particularly different because only the opposite applications in terms of loading were analyzed. The results show that the p-values are lower than 0.05, which means the approaches are significantly different. However, as the results indicate, the Drift-enhanced mechanism adds some computational costs to the edge device. It means the difference between them states the policy-based mechanism outperformed in this scenario.

Table 19 – Mann-Whitney U test results for edge device metrics. It compares two offloading strategies: policy-based and Drift-enhanced.

Application	Metric	Statistics	p-value
ddos-10s (750)	CPU	12864248.000	0.000
	Memory	13054339.000	0.000
	Bandwidth	4972451.500	0.000
	Latency	1370138.500	0.695
ddos-1s (250)	CPU	12696392.000	0.000
	Memory	16805827.000	0.000
	Bandwidth	2019870.000	0.000
	Latency	3488019.000	0.000

Source: Author

4.2.11 Conclusion

This section described the performance evaluation of DAOS, responsible for comparing the intelligence brought by it with a policy-based offloading mechanism. The instance of DAOS is an experimental software implemented to be evaluated in different environments (edge, cloud), serving as an infrastructure for executing applications in the context of Intrusion Detection Systems (IDS). The results demonstrate that DAOS is suitable for being deployed as an edge infrastructure system that offloads CEP applications to detect DDoS attacks against IoT devices. As shown in section 4.2.9.2, it presents improvements in the edge devices' overall performance compared to the policy-based mechanism.

Despite the attacker sending a higher rate of malicious packets in the DDoS detection case, the intelligent offloading mechanism would optimize the processing transfer to the cloud. Consequently, this optimization ends up minimizing the latency and resource usage of CEP applications in the edge, enabling the network's security administrator to take an earlier action when receiving a DDoS attack notification. As a result, this improvement opens a door for using DAOS and CEP to support time-sensitive applications that run close to the data sources, such as fraud detection, predictive maintenance, and fall detection systems. Moreover, considering the computation cost of adding DAOS in the infrastructure layer, it may not be suitable for supporting applications that do not require a timely response.

The beneficial results for adopting DAOS were statistically compared with the alternative approach (policy-based), showing a significant difference. Moreover, the Figures 20 and 23 visually demonstrated that DAOS effectively starts the offloading even before the policy-based mechanism take the same action. It also clarifies the trade-off between both approaches since the policy-based can also be optimized to offload in a better timing but requires manual intervention. On the other hand, the results show that the adoption of concept drift detection effectively increases the adaptability of DAOS.

Figure 23 reveals an unrealistic scenario in which multiple offloading occurred over time. This behavior was intentionally introduced to vary the underlying profiling data, leading to concept drift occurrences. Therefore, the main limitation of the Drift-enhanced evaluation was the absence of a good testbed for reproducing the drifts. Despite this limitation, the comparison between the Figures 23a and 23b is relevant to indicate the increase in process from one to another, resulting in the subsequent offloading. Figure 23b shows a scenario where the fallback mechanism starts using the policy-based strategy after a concept drift occurrence. The last offloading marked in the chart is the successful proof of the effectiveness of this mechanism, which does not compromise the offloading performance even facing a drift.

5 RELATED WORKS

This chapter discusses the works related to the research, pointing out their contributions and drawbacks. We present a non-exhaustive list of related work that resulted from an exploratory literature review. The selection criteria were the number of citations and their similarity to the explored problem.

5.1 OPERATOR PLACEMENT

In CEP terminology, the operator is a functional computing unit that represents event processing capabilities such as filtering and aggregation (LUTHRA, 2018). In order to deliver faster response, the operators must be placed in a way that minimizes factors such as latency, response time, and energy consumption. The following works proposed solutions to solve this placement problem:

(CAI et al., 2018) proposes a response time aware strategy for operator placement in edge computing. It presents an approximation-based algorithm that considers the response time as the most important Quality of Service (QoS) metric for optimizing the placement decision. Unlike DAOS, it focuses on distributing the operators in the network topology and uses optimization techniques to solve a NP-hard problem.

TCEP (LUTHRA et al., 2021) is a transition-capable CEP system that supports operator placement transitions. It uses a lightweight online genetic algorithm to meet the QoS requirements and decide the placement of some operators. This solution presents similarities to DAOS regarding the online learning aspect of the generic algorithm. However, rather than selecting the operator's placement, the focus of DAOS is deciding the best moment to offload a CEP application to the cloud.

Therefore, operator placement solutions generally focus on answering the *where* offloading question, as explained in section 2.2.

5.2 STATIC OFFLOADING

In the computation offloading literature, some solutions identify and migrate computing-intensive code to meet QoS requirements (FLORES et al., 2015) (CHEN et al., 2019). The traditional mechanism to enable code offloading are context-aware and use limits or timeouts to control when the offloading should happen. It can consider any other offloadable workload as an offloading unit, such as the event streaming in the CEP offloading scenario. Some approaches that use this type of static offloading decision are listed below:

mCloud (ZHOU et al., 2017) is a code offloading framework that uses a context-aware decision algorithm to offload mobile applications. The architecture of mCloud is

similar to DAOS, but it uses a general cost estimation model for helping in offloading decisions with no ability to learn with the contextual data.

GiTo (FONSECA; FERRAZ; GAMA, 2018) is a distributed CEP coordination system that uses a policy-based mechanism to decide when offloading Web of Things (WoT) applications. It distributes the processing between the mist, fog, and cloud layers, where a set of policies are defined based on resource usage thresholds that head the offload decision between the layers. On the contrary, DAOS proposes an intelligent mechanism that uses online learning to decide when offloading CEP applications in the edge. Nevertheless, the policy-based mechanism of GiTo was adopted by DAOS to be used as a fallback strategy when the model is not reliable and serves as a baseline when comparing the intelligent approach to the policy-based one.

5.3 INTELLIGENT OFFLOADING

Recently, as an alternative to the static offloading mechanisms, some works are using machine learning to build intelligent decision algorithms that can learn the best policies over time (CAO et al., 2019) (SHAKARAMI; SHAHIDINEJAD; GHOBAEI-ARANI, 2020). Moreover, there are works that apply this adaptive strategies to the context of edge computing, which approximate them to WoT scenarios (CARVALHO et al., 2020) (HOSSAIN et al., 2020) (WANG et al., 2021). Given the limited available resources of edge devices, it can be hard to train traditional machine learning algorithms. Consequently, there are also works adopting the online learning models (XU; CHEN; REN, 2017) (SUN et al., 2019).

MALMOS (EOM et al., 2015) is a framework for mobile offloading that uses online machine learning for making accurate scheduling decisions. It verifies the correctness of the previous offloading decisions and use it as an input to improve the current model. Despite their similar architecture, there is a considerable difference between MALMOS and DAOS regarding the used algorithms and the training strategy. In DAOS, the online learning algorithm is adapted for a data streaming scenario. Moreover, the training is continuous, without being scheduled as in MALMOS.

(REGO et al., 2017) proposes a solution that uses decision trees in the offloading decision process. The trees are created through the offloading previously performed and the profiling information, giving a binary output (local or remote), similar to the binary classification adopted by DAOS. The difference between them is that the decision trees are created in the cloud and then synchronized with the mobile devices without supporting online learning. In addition, it aims to minimize energy consumption in the mobile context, which is different from the scenarios approached by DAOS.

(JUNIOR et al., 2019) proposes the Context-Sensitive Offloading System (CSOS), which is a solution for improving the accuracy in offloading mobile applications through the usage of contextual information. It was possible with the application of machine learning algorithms such as KNN, Decision Tree, Rules, and Naive Bayes. The models

were previously trained with a dataset of contextual data and then used on the offloading decision process. It has similarities with DAOS regarding the architecture and the machine learning usage but focuses on integrating the decision mechanism with Software-Defined Networking (SDN) infrastructure and has no support for online learning.

5.4 SUMMARY

This chapter discussed works that have something in common with DAOS. Firstly, there are initiatives for CEP operator placement, but the optimization solutions do not necessarily focus on answering the *when* offloading question. Secondly, some works propose offloading decision systems that use static indicators to drive the decision, such as resource thresholds (policy-based) or timeouts. Machine learning algorithms come as a solution for making this decision more dynamic. However, the constrained environment of edge computing imposes challenges to using memory-consuming algorithms, and most of the works do not use algorithms for data streaming settings.

The main contribution of DAOS is using online learning and drift detection to decide *when* to offload CEP workload in the edge. This research did not find works that use concept drift detection for tasks other than adjusting the machine learning models. It opens an opportunity for using concept drift alerts to adapt to changes in the profiling data, considering its influence on the offloading decision. The main advantage of considering concept drift is that it reflects how resource usage evolves when facing contextual changes. Therefore, DAOS uses those occurrences to change between the intelligent and the static approaches via fallback, guaranteeing reliability when the models are not safe.

6 CONCLUSION

Edge computing has been predominantly leveraging the applications that require low latency in critical scenarios. In a complementary way, the adoption of CEP is fundamental in a world that handles millions of events per second from those scenarios. Both paradigms can work together to deliver value to the clients, meeting the QoS requirements. Nevertheless, CEP engines usually require a machine with an excellent capacity to process multiple data streams, which is not always the case when talking about edge devices. Despite this challenge, computation offloading strategies help divide the processing between multiple devices when the edge becomes overloaded. One of the most crucial questions that offloading techniques try to answer is: **when to offload?**

This research presented a drift adaptive offloading system (DAOS) that introduces a way for adapting offloading systems to changes in the environment. It uses online machine learning to answer the *when* question, taking into consideration the occurrence of concept drifts in the profiling data. In addition, it evaluates DAOS by comparing it to the policy-based solutions that use static thresholds to answer the *when* question. The obtained results show that the difference between them is significant. It has demonstrated that the improvements introduced by DAOS enhanced the performance of edge devices when offloading the CEP application responsible for detecting DDoS attacks, responding to the first research question (**Q1**) about the feasibility of using online learning techniques in a fast and efficient way.

Moreover, the experiment reveals that taking offloading decisions with the Drift-enhanced mechanism and the fallback mechanism does not negatively impact the performance. In the executions involving this approach, the time was increased from 30 to 120 minutes to detect concept drifts on the profiling events more easily. The fallback mechanism has interchangeably used the policy-based and ML-enhanced strategies. Consequently, even with a reduction in the prediction reliability, the decision mechanism has successfully fallback to use policies. It naturally prevents the edge device from maintaining a high resource consumption for a long time, responding to the second research question (**Q2**) about the increase in effectiveness and reliability of the offloading process.

6.1 CONTRIBUTIONS

We believe this work brings important evaluations that can support the choice of a more appropriate offloading approach for different types of time-sensitive Complex Event Processing applications. More specifically, the main contributions of this research are:

1. Comparing static and dynamic (i.e. intelligent) offloading decision approaches in the context of CEP.

2. Evaluating different online learning algorithms to select the most appropriate for the offloading problem.

3. Using online learning algorithms suitable for data streaming scenarios to help in the offloading decision process.

4. Using concept drift detection not only to retrain the online learning models but also guide the decision algorithm to use policies until the models become reliable again.

The initial results of this research were published as an extended abstract in the ACM/IEEE Symposium on Edge Computing (NETO; FONSECA; GAMA, 2020). Moreover, the source code of DAOS and the experimental scripts are publicly available in GitHub[1].

6.2 FUTURE WORKS

- **Reinforcement Learning (RL)**: There is a recent trend in designing offloading techniques that use RL as an intelligent decision mechanism (WANG et al., 2021) (SHAKARAMI; SHAHIDINEJAD; GHOBAEI-ARANI, 2020). According to (KAELBLING; LITTMAN; MOORE, 1996), RL describes the problem of an agent learning a new behavior via trial-and-error interactions. Depending on the agent's decisions, it gets a reward or a penalty, in which the total reward is the maximization goal. In the offloading context, the agent is an algorithm trying to learn the best decision by interacting with the environment (HOSSAIN et al., 2020). For instance, the algorithm running in an edge device can decide to offload when the CPU consumption reaches 70%, getting a reward depending on whether the offload improves some metrics, such as energy consumption and latency.

- **QoS-driven Offloading**: This research focused on evaluating DAOS with a device's performance point-of-view. Another aspect that can be explored is the usage of QoS metrics, such as latency and response time, as a parameter that should be optimized during the offloading decision. This evaluation would be important for validating how it affects the experience of applications that depends on the CEP responses. The usage of QoS metrics can also fit together with Reinforcement Learning algorithms, being used as an indicator to reward or penalty the algorithm.

- **Offloading Assessment**: The online learning model used in DAOS is assessed through the output of the policy-based mechanism. However, this can negatively affect the results since the algorithm learns with a static approach that only considers metrics from the device and the network, which is similar to not using QoS metrics. Based on this, a future work can be assessing the results with metrics that indicate whether the offloading done previously improved or not the overall performance.

[1] https://github.com/netoax/adaptive-offloading-system

Some of these metrics are formulated by (SHAKARAMI; SHAHIDINEJAD; GHOBAEI-ARANI, 2020), such as energy consumption, delay, cost of offloading, and provider's profit. Moreover, the Hoeffing Tree model and others in data streaming settings can deal with delayed assessments, where offloading results arrive later.

- **Extension**: The architecture of DAOS is not limited to CEP applications. It was conceived independently from the CEP engines, being adaptable for other resource-intensive applications in the edge that also need offloading. Therefore, this extension of DAOS to different domains can be an opportunity for future works.

REFERENCES

ABOWD, G. D.; DEY, A. K.; BROWN, P. J.; DAVIES, N.; SMITH, M.; STEGGLES, P. Towards a Better Understanding of Context and Context-Awareness. In: *International symposium on handheld and ubiquitous computing*. [s.n.], 1999. p. 304–307. ISBN 978-3-540-48157-7. Disponível em: <http://link.springer.com/10.1007/3-540-48157-5_29>.

ADI, A.; BOTZER, D.; NECHUSHTAI, G.; SHARON, G. Complex event processing for financial services. *Proceedings - SCW 2006: IEEE Services Computing Workshops*, p. 7–12, 2006.

AGRAWAL, D.; CALO, S.; LEE, K.-w.; LOBO, J.; VERMA, D. *Policy technologies for self-managing systems*. [S.l.: s.n.], 2008. ISBN 978-0-13-221307-3.

AGRAWAL, J.; DIAO, Y.; GYLLSTROM, D.; IMMERMAN, N. Efficient pattern matching over event streams. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, p. 147–159, 2008. ISSN 07308078.

ALAM, M. G. R.; HASSAN, M. M.; UDDIN, M. Z.; ALMOGREN, A.; FORTINO, G. Autonomic computation offloading in mobile edge for IoT applications. *Future Generation Computer Systems*, Elsevier B.V., v. 90, p. 149–157, 2019. ISSN 0167739X. Disponível em: <https://doi.org/10.1016/j.future.2018.07.050>.

BABCOCK, B.; BABU, S.; DATAR, M.; MOTWANI, R.; WIDOM, J. Models and issues in data stream systems. p. 1, 2002.

BAKSHI, K. Considerations for big data: Architecture and approach. *IEEE Aerospace Conference Proceedings*, p. 1–7, 2012. ISSN 1095323X.

BARROS, R. S. M. d.; HIDALGO, J. I. G.; CABRAL, D. R. d. L. Wilcoxon Rank Sum Test Drift Detector. *Neurocomputing*, Elsevier, v. 275, p. 1954–1963, 1 2018. ISSN 0925-2312.

BASILI, V. R.; CALDIERA, G. The Goal Question Metric Paradigm. *Encyclopedia of Software Engineering - 2 Volume Set*, v. 2, p. 528–532, 1994. Disponível em: <https://www.cs.umd.edu/~basili/publications/technical/T89.pdf>.

BENCZÚR, A. A.; KOCSIS, L.; PÁLOVICS, R. Online Machine Learning in Big Data Streams: Overview. *Encyclopedia of Big Data Technologies*, p. 1207–1218, 2019.

BERGER, J. O. *Statistical Decision Theory and Bayesian Analysis*. New York, NY: Springer New York, 1985. (Springer Series in Statistics). ISBN 978-1-4419-3074-3. Disponível em: <http://link.springer.com/10.1007/978-1-4757-4286-2>.

BIFET, A.; GAVALDÀ, R. Learning from time-changing data with adaptive windowing. *Proceedings of the 7th SIAM International Conference on Data Mining*, p. 443–448, 2007.

BIFET, A.; KIRKBY, R. DATA STREAM MINING A Practical Approach. In: . [S.l.: s.n.], 2009.

BIFET, A.; MORALES, G. de F.; READ, J.; HOLMES, G.; PFAHRINGER, B. Efficient Online Evaluation of Big Data Stream Classifiers. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2015. v. 2015-Augus, p. 59–68. ISBN 9781450336642. Disponível em: <https://dl.acm.org/doi/10.1145/2783258.2783372>.

BIFET, A.; ZHANG, J.; FAN, W.; HE, C.; ZHANG, J.; QIAN, J.; HOLMES, G.; PFAHRINGER, B. Extremely fast decision tree mining for evolving data streams. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Part F1296, p. 1733–1742, 2017.

BUTAKOVA, M. A.; CHERNOV, A. V.; SHEVCHUK, P. S.; VERESKUN, V. D. Complex event processing for network anomaly detection in digital railway communication services. *2017 25th Telecommunications Forum, TELFOR 2017 - Proceedings*, v. 2017-Janua, p. 1–4, 2018.

CABRAL, D. R. d. L.; BARROS, R. S. M. d. Concept drift detection based on Fisher's Exact test. *Information Sciences*, Elsevier, v. 442-443, p. 220–234, 5 2018. ISSN 0020-0255.

CAI, X.; KUANG, H.; HU, H.; SONG, W.; LÜ, J. Response Time Aware Operator Placement for Complex Event Processing in Edge Computing. In: . Springer International Publishing, 2018. v. 1, p. 264–278. ISBN 9783030035969. Disponível em: <https://doi.org/10.1007/978-3-030-03596-9_18http://link.springer.com/10.1007/978-3-030-03596-9_18>.

CAO, B.; ZHANG, L.; LI, Y.; FENG, D.; CAO, W. Intelligent Offloading in Multi-Access Edge Computing: A State-of-the-Art Review and Framework. *IEEE Communications Magazine*, IEEE, v. 57, n. 3, p. 56–62, 2019. ISSN 15581896.

CARVALHO, G.; CABRAL, B.; PEREIRA, V.; BERNARDINO, J. Computation offloading in Edge Computing environments using Artificial Intelligence techniques. *Engineering Applications of Artificial Intelligence*, Elsevier Ltd, v. 95, n. April, p. 103840, 2020. ISSN 09521976. Disponível em: <https://doi.org/10.1016/j.engappai.2020.103840>.

CHEN, J.; CHEN, C. Design of complex event-processing IDS in internet of things. *Proceedings - 2014 6th International Conference on Measuring Technology and Mechatronics Automation, ICMTMA 2014*, p. 226–229, 2014.

CHEN, X.; CHEN, S.; MA, Y.; LIU, B.; ZHANG, Y.; HUANG, G. An adaptive offloading framework for Android applications in mobile edge computing. *Science China Information Sciences*, v. 62, n. 8, p. 82102, 8 2019. ISSN 1674-733X. Disponível em: <http://link.springer.com/10.1007/s11432-018-9749-8>.

Constantinos Kolias; Georgios Kambourakis; Angelos Stavrou; Jeffrey Voas. DDoS in the IoT: Mirai and Other Botnets. *Computer*, p. 1, 2017.

CUGOLA, G.; MARGARA, A. Processing flows of information: From data stream to complex event processing. *ACM Computing Surveys*, v. 44, n. 3, 2012. ISSN 03600300.

CUGOLA, G.; MARGARA, A.; MATTEUCCI, M.; TAMBURRELLI, G. *Introducing uncertainty in complex event processing: model, implementation, and validation*. [S.l.: s.n.], 2015. v. 97. 103–144 p. ISSN 14365057. ISBN 0060701404.

DATTATREYA, G. R. Decision trees. *Artificial Intelligence Methods in the Environmental Sciences*, p. 77–101, 2009.

DAYARATHNA, M.; PERERA, S. Recent advancements in event processing. *ACM Computing Surveys*, v. 51, n. 2, 2018. ISSN 15577341.

DIETTERICH, T. G. Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. *Neural Computation*, MIT Press, v. 10, n. 7, p. 1895–1923, 10 1998. ISSN 0899-7667.

DILLON, T.; WU, C.; CHANG, E. Cloud computing: Issues and challenges. *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, p. 27–33, 2010. ISSN 1550445X.

DOMINGOS, P.; HULTEN, G. Mining high-speed data streams. *Proceeding of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Association for Computing Machinery (ACM), p. 71–80, 2000.

EOM, H.; FIGUEIREDO, R.; CAI, H.; ZHANG, Y.; HUANG, G. MALMOS: Machine learning-based mobile offloading scheduler with online training. *Proceedings - 2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, MobileCloud 2015*, p. 51–60, 2015.

ETZION, O.; NIBLETT, P. *Event Processing In action*. [S.l.: s.n.], 2011. 386 p. ISBN 9781935182214.

FLORES, H.; HUI, P.; TARKOMA, S.; LI, Y.; SRIRAMA, S.; BUYYA, R. Mobile code offloading: From concept to practice and beyond. *IEEE Communications Magazine*, v. 53, n. 3, p. 80–88, 2015. ISSN 01636804.

FONSECA, J.; FERRAZ, C.; GAMA, K. Doctoral symposium: A policy-based coordination architecture for distributed complex event processing in the internet of things. *DEBS 2016 - Proceedings of the 10th ACM International Conference on Distributed and Event-Based Systems*, p. 418–421, 2016.

FONSECA, J.; FERRAZ, C.; GAMA, K. Migrating complex event processing in the Web of Things. In: *WebMedia 2018 - Proceedings of the 24th Brazilian Symposium on Multimedia and the Web*. [S.l.: s.n.], 2018. ISBN 9781450358675.

FORMAN, G. H.; ZAHORJAN, J. The Challenges of Mobile Computing. *Computer*, v. 27, n. 4, p. 38–47, 1994. ISSN 00189162.

GABER, M. M.; ZASLAVSKY, A.; KRISHNASWAMY, S. Data Stream Mining. *Data Mining and Knowledge Discovery Handbook*, n. May, p. 759–787, 2009.

GAMA, J.; MEDAS, P.; ROCHA, R. Forest trees for on-line data. *Proceedings of the ACM Symposium on Applied Computing*, v. 1, p. 632–636, 2004.

GAMA, J.; ROCHA, R.; MEDAS, P. Accurate decision trees for mining high-speed data streams. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, p. 523–528, 2003.

GAMA, J.; SEBASTIÃO, R.; RODRIGUES, P. P. Issues in evaluation of stream learning algorithms. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, p. 329–337, 2009.

GAMA, J.; SEBASTIÃO, R.; RODRIGUES, P. P. On evaluating stream learning algorithms. *Machine Learning*, v. 90, n. 3, p. 317–346, 2013. ISSN 08856125.

GAMA, J.; ŽLIOBAITĖ, I.; BIFET, A.; PECHENIZKIY, M.; BOUCHACHIA, A. A survey on concept drift adaptation. *ACM Computing Surveys*, v. 46, n. 4, p. 1–37, 4 2014. ISSN 0360-0300. Disponível em: <http://www.eurekaselect.com/openurl/content.php?genre=article&issn=1570-1646&volume=7&issue=4&spage=258https://dl.acm.org/doi/10.1145/2523813>.

GONG, C.; LIU, J.; ZHANG, Q.; CHEN, H.; GONG, Z. The characteristics of cloud computing. *Proceedings of the International Conference on Parallel Processing Workshops*, p. 275–279, 2010. ISSN 15302016.

GRULICH, P. M.; SAITENMACHER, R.; BRESS, S.; RABL, T.; TRAUB, J.; MARKL, V. Scalable detection of concept drifts on data streams with parallel adaptive windowing. *Advances in Database Technology - EDBT*, v. 2018-March, p. 477–480, 2018. ISSN 23672005.

GU, F.; ZHANG, G.; LU, J.; LIN, C. T. Concept drift detection based on equal density estimation. *Proceedings of the International Joint Conference on Neural Networks*, Institute of Electrical and Electronics Engineers Inc., v. 2016-October, p. 24–30, 10 2016.

GUMUS, F.; SAKAR, C. O.; ERDEM, Z.; KURSUN, O. Online Naive Bayes classification for network intrusion detection. *ASONAM 2014 - Proceedings of the 2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, Institute of Electrical and Electronics Engineers Inc., p. 670–674, 10 2014.

GUO, G.; WANG, H.; BELL, D.; BI, Y.; GREER, K. KNN model-based approach in classification. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 2888, p. 986–996, 2003. ISSN 16113349.

GUO, H.; LIU, J.; LV, J. Toward Intelligent Task Offloading at the Edge. *IEEE Network*, IEEE, PP, n. Ml, p. 1–7, 2019. ISSN 1558156X.

HAENLEIN, M.; KAPLAN, A. A brief history of artificial intelligence: On the past, present, and future of artificial intelligence. *California Management Review*, v. 61, n. 4, p. 5–14, 2019. ISSN 21628564.

HAGHIGHI, V.; MOAYEDIAN, N. S. An offloading strategy in mobile cloud computing considering energy and delay constraints. *IEEE Access*, v. 6, n. c, p. 11849–11861, 2018. ISSN 21693536.

HASHEM, I. A. T.; YAQOOB, I.; ANUAR, N. B.; MOKHTAR, S.; GANI, A.; KHAN, S. U. The rise of "big data" on cloud computing: Review and open research issues. *Information Systems*, Elsevier, v. 47, p. 98–115, 2015. ISSN 03064379. Disponível em: <http://dx.doi.org/10.1016/j.is.2014.07.006>.

HOEFFDING, W. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, v. 58, n. 301, p. 13–30, 1963. ISSN 1537274X.

HOSSAIN, M. S.; NWAKANMA, C. I.; LEE, J. M.; KIM, D. S. Edge computational task offloading scheme using reinforcement learning for IIoT scenario. *ICT Express*, Elsevier B.V., v. 6, n. 4, p. 291–299, 2020. ISSN 24059595. Disponível em: <https://doi.org/10.1016/j.icte.2020.06.002>.

JAIN, R. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. [S.l.]: John Wiley \& Sons, 1990.

JIANG, C.; CHENG, X.; GAO, H.; ZHOU, X.; WAN, J. Toward Computation Offloading in Edge Computing: A Survey. *IEEE Access*, IEEE, v. 7, p. 131543–131558, 2019. ISSN 21693536.

JUNIOR, W.; OLIVEIRA, E.; SANTOS, A.; DIAS, K. A context-sensitive offloading system using machine-learning classification algorithms for mobile cloud environment. *Future Generation Computer Systems*, North-Holland, v. 90, p. 503–520, 1 2019. ISSN 0167-739X.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, v. 4, n. 19, p. 237–285, 5 1996. ISSN 1076-9757. Disponível em: <https://www.jair.org/index.php/jair/article/view/10166>.

KHAN, A. u. R.; OTHMAN, M.; MADANI, S. A.; KHAN, S. U. A Survey of Mobile Cloud Computing Application Models. *IEEE Communications Surveys & Tutorials*, v. 16, n. 1, p. 393–413, 2014. ISSN 1553-877X. Disponível em: <https://ieeexplore.ieee.org/document/6553297/>.

KORONIOTIS, N.; MOUSTAFA, N.; SITNIKOVA, E.; TURNBULL, B. Towards the development of realistic botnet dataset in the Internet of Things for network forensic analytics: Bot-IoT dataset. *Future Generation Computer Systems*, v. 100, p. 779–796, 2019. ISSN 0167739X.

KUMAR, K.; LIU, J.; LU, Y. H.; BHARGAVA, B. A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, v. 18, n. 1, p. 129–140, 2013. ISSN 1383469X.

KWAK, S. G.; KIM, J. H. Central limit theorem: the cornerstone of modern statistics. *Korean Journal of Anesthesiology*, Korean Society of Anesthesiologists, v. 70, n. 2, p. 144, 4 2017. ISSN 20057563. Disponível em: </pmc/articles/PMC5370305//pmc/articles/PMC5370305/?report=abstracthttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC5370305/>.

LANEY, D. 3D data management: Controlling data volume, velocity and variety. *META group research note*, Stanford, v. 6, n. 70, p. 1, 2001.

LAW, Y. N.; ZANIOLO, C. An adaptive nearest neighbor classification algorithm for data Streams. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 3721 LNAI, p. 108–120, 2005. ISSN 03029743.

LEWIS, R. J. An Introduction to Classification and Regression Tree (CART) Analysis. *Annual meeting of the society for academic emergency medicine in San Francisco, California*, v. 14, 2000.

LIU, A. C Oncept D Rift a Daptation for L Earning With S Treaming. n. April, 2018.

LIU, A.; SONG, Y.; ZHANG, G.; LU, J. Regional Concept Drift Detection and Density Synchronized Drift Adaptation. 2017.

LIU, A.; ZHANG, G.; LU, J. Fuzzy time windowing for gradual concept drift adaptation. *IEEE International Conference on Fuzzy Systems*, Institute of Electrical and Electronics Engineers Inc., 8 2017. ISSN 10987584.

LIU, F.; TANG, G.; LI, Y.; CAI, Z.; ZHANG, X.; ZHOU, T. A Survey on Edge Computing Systems and Tools. *Proceedings of the IEEE*, v. 107, n. 8, 2019. ISSN 00189219.

LU, J.; LIU, A.; DONG, F.; GU, F.; GAMA, J.; ZHANG, G. Learning under Concept Drift: A Review. *IEEE Transactions on Knowledge and Data Engineering*, v. 31, n. 12, p. 2346–2363, 2019. ISSN 15582191.

LUCKHAM, D. C.; FRASCA, B. Complex event processing in distributed systems. *... Systems Laboratory Technical Report CSL- ...*, 1998. Disponível em: <http://www.cs.rmit.edu.au/agents/www/meetings/agentgroupmeetings_material/readings/carlosq-24jan08-luckham98complex.pdf>.

LUTHRA, M. Adapting to Dynamic User Environments in Complex Event Processing System using Transitions. p. 274–277, 2018.

LUTHRA, M.; KOLDEHOFE, B.; DANGER, N.; WEISENBERGER, P.; SALVANESCHI, G.; STAVRAKAKIS, I. TCEP: Transitions in operator placement to adapt to dynamic network environments. *Journal of Computer and System Sciences*, Elsevier Inc., v. 122, p. 94–125, 2021. ISSN 10902724. Disponível em: <https://doi.org/10.1016/j.jcss.2021.05.003>.

MACH, P.; BECVAR, Z. Mobile Edge Computing: A Survey on Architecture and Computation Offloading. *IEEE Communications Surveys and Tutorials*, v. 19, n. 3, p. 1628–1656, 2017. ISSN 1553877X.

MANAPRAGADA, C.; WEBB, G. I.; SALEHI, M. Extremely fast decision tree. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, p. 1953–1962, 2018.

MANN, H. B.; WHITNEY, D. R. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *https://doi.org/10.1214/aoms/1177730491*, Institute of Mathematical Statistics, v. 18, n. 1, p. 50–60, 3 1947. ISSN 0003-4851. Disponível em: <https://projecteuclid.org/journals/annals-of-mathematical-statistics/volume-18/issue-1/On-a-Test-of-Whether-one-of-Two-Random-Variables/10.1214/aoms/

1177730491.fullhttps://projecteuclid.org/journals/annals-of-mathematical-statistics/volume-18/issue-1/On-a-Test-of-Whether-one-of-Two-Random-Variables/10.1214/aoms/1177730491.short>.

MORENO, A. M. BASICS OF SOFTWARE Natalia Juristo. *Analysis*, p. 367, 2001.

MURICY, W.; JUNIOR, V. A Context-sensitive offloading system using machine-learning classification algorithms with seamless mobility support. 2018.

MYLES, A. J.; FEUDALE, R. N.; LIU, Y.; WOODY, N. A.; BROWN, S. D. An introduction to decision tree modeling. *Journal of Chemometrics*, v. 18, n. 6, p. 275–285, 2004. ISSN 08869383.

NETO, J. A.; FONSECA, J. C.; GAMA, K. Towards Online Learning and Concept Drift for Offloading Complex Event Processing in the Edge. *Proceedings - 2020 IEEE/ACM Symposium on Edge Computing, SEC 2020*, Institute of Electrical and Electronics Engineers Inc., p. 167–169, 11 2020.

REGO, P. A.; CHEONG, E.; COUTINHO, E. F.; TRINTA, F. A.; HASANY, M. Z.; SOUZA, J. N. Decision Tree-Based Approaches for Handling Offloading Decisions and Performing Adaptive Monitoring in MCC Systems. *Proceedings - 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, MobileCloud 2017*, p. 74–81, 2017.

RISH, I. et al. An empirical study of the naive Bayes classifier. In: *IJCAI 2001 workshop on empirical methods in artificial intelligence*. [S.l.: s.n.], 2001. v. 3, p. 41–46.

ROKACH, L.; MAIMON, O. Z. *Data mining with decision trees: theory and applications*. [S.l.: s.n.], 2007. ISBN 9789814590075.

SARITAS, M. M.; YASAR, A. Performance Analysis of ANN and Naive Bayes Classification Algorithm for Data Classification. *International Journal of Intelligent Systems and Applications in Engineering*, v. 7, n. 2, p. 88–91, 6 2019. ISSN 2147-6799. Disponível em: <https://ijisae.org/IJISAE/article/view/934>.

SATYANARAYANAN, M. The emergence of edge computing. *Computer*, v. 50, n. 1, p. 30–39, 2017. ISSN 00189162.

SHAKARAMI, A.; SHAHIDINEJAD, A.; GHOBAEI-ARANI, M. A review on the computation offloading approaches in mobile edge computing: A game-theoretic perspective. *Software - Practice and Experience*, v. 50, n. 9, p. 1719–1759, 2020. ISSN 1097024X.

SHAPIRO, S. S.; WILK, M. B. An analysis of variance test for normality (complete samples). *Biometrika*, Oxford University Press (OUP), v. 52, n. 3-4, p. 591–611, 12 1965. ISSN 0006-3444.

SHI, W.; CAO, J.; ZHANG, Q.; LI, Y.; XU, L. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, v. 3, n. 5, p. 637–646, 2016. ISSN 23274662.

SHI, W.; DUSTDAR, S. CLOUD COVER WHY DO WE NEED EDGE COMPUTING? The Promise of Edge Computing. n. 0018, 2016.

SUN, Y.; GUO, X.; SONG, J.; ZHOU, S.; JIANG, Z.; LIU, X.; NIU, Z. Adaptive Learning-Based Task Offloading for Vehicular Edge Computing Systems. *IEEE Transactions on Vehicular Technology*, IEEE, v. 68, n. 4, p. 3061–3074, 2019. ISSN 19399359.

TCL, A. AGENT TCL : the Needs of. p. 58–67, 1997.

TSYMBAL, A. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, n. May, 2004.

VINAGRE, J.; JORGE, A. M.; ROCHA, C.; GAMA, J. Statistically Robust Evaluation of Stream-Based Recommender Systems. *IEEE Transactions on Knowledge and Data Engineering*, v. 33, n. 7, p. 2971–2982, 7 2021. ISSN 1041-4347. Disponível em: <https://ieeexplore.ieee.org/document/8935156/>.

VOJTˇ, V.; HORKÝ, V.; LIBIČ, P.; STEINHAUSER, A.; UMA, P. T. DOs and DON'Ts of Conducting Performance Measurements in Java. Disponível em: <http://dx.doi.org/10.1145/2668930.2688820.>

WANG, J.; HU, J.; MIN, G.; ZOMAYA, A. Y.; GEORGALAS, N. Fast Adaptive Task Offloading in Edge Computing Based on Meta Reinforcement Learning. *IEEE Transactions on Parallel and Distributed Systems*, v. 32, n. 1, p. 242–253, 2021. ISSN 15582183.

WANG, J.; TEPFENHART, W. *Formal Methods in Computer Science*. Boca Raton : Taylor & Francis, a CRC title, part of the Taylor & Francis imprint, a member of the Taylor & Francis Group, the academic division of T&F Informa, plc, 2019.: Chapman and Hall/CRC, 2019. ISBN 9780429184185. Disponível em: <https://www.taylorfrancis.com/books/9781498775335>.

WEBB, G. I. Encyclopedia of Machine Learning and Data Mining. *Encyclopedia of Machine Learning and Data Mining*, n. January 2016, 2016.

WOHLIN, C.; RUNESON, P.; HÖST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLÉN, A. Experimentation in software engineering. *Experimentation in Software Engineering*, Springer-Verlag Berlin Heidelberg, v. 9783642290442, p. 1–236, 7 2012.

WOOD, A.; SHPILRAIN, V.; NAJARIAN, K.; KAHROBAEI, D. Private naive bayes classification of personal biomedical data: Application in cancer data analysis. *Computers in Biology and Medicine*, Pergamon, v. 105, p. 144–150, 2 2019. ISSN 0010-4825.

WU, Q.; LIN, H.; JIN, Y.; CHEN, Z.; LI, S.; CHEN, D. A new fallback beetle antennae search algorithm for path planning of mobile robots with collision-free capability. *Soft Computing*, Springer Berlin Heidelberg, v. 24, n. 3, p. 2369–2380, 2020. ISSN 14337479. Disponível em: <https://doi.org/10.1007/s00500-019-04067-3>.

XIAO, Y.; LIU, J.; GHABOOSI, K.; DENG, H.; ZHANG, J. Botnet: Classification, attacks, detection, tracing, and preventive measures. *Eurasip Journal on Wireless Communications and Networking*, v. 2009, 2009. ISSN 16871472.

XU, J.; CHEN, L.; REN, S. Online Learning for Offloading and Autoscaling in Energy Harvesting Mobile Edge Computing. *IEEE Transactions on Cognitive Communications and Networking*, v. 3, n. 3, p. 361–373, 2017. ISSN 23327731.

XU, S.; WANG, J. Dynamic extreme learning machine for data stream classification. *Neurocomputing*, Elsevier, v. 238, p. 433–449, 5 2017. ISSN 0925-2312.

YANG, C.; HUANG, Q.; LI, Z.; LIU, K.; HU, F. Big Data and cloud computing: innovation opportunities and challenges. *International Journal of Digital Earth*, Taylor & Francis, v. 10, n. 1, p. 13–53, 2017. ISSN 17538955.

YU, S.; WANG, X.; LANGAR, R. Computation offloading for mobile edge computing: A deep learning approach. *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC*, v. 2017-Octob, p. 1–6, 2018.

ZHOU, B.; DASTJERDI, A. V.; CALHEIROS, R. N.; SRIRAMA, S. N.; BUYYA, R. mCloud: A Context-Aware Offloading Framework for Heterogeneous Mobile Cloud. *IEEE Transactions on Services Computing*, v. 10, n. 5, p. 797–810, 9 2017. ISSN 1939-1374. Disponível em: <http://ieeexplore.ieee.org/document/7362223/>.