# TOWARDS MACHINE LEARNING BASED ACCESS CONTROL

by

MOHAMMAD NUR NOBI, M.Sc.

DISSERTATION
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Sciences
Department of Computer Science
August 2022

## DEDICATION

*I want to dedicate this dissertation to my parents, Mr. Mohammad Tajul Islam and Mrs. Shamsun Naher, for their amazing support, unconditional love, and prayers. Also, I would like to dedicate it to my brother Imam, sister Hazera, and my deceased brother Masud, who always inspired me in every step of my life. Finally, I dedicate this piece to my beloved wife, Sonia, and my lovely princess, Nuzhat, who are my source of strength.*

# ACKNOWLEDGEMENTS

To begin with, I am praising the Almighty for the guidance, strength, power of the mind, skills, and after all, for giving me a healthful life to continue this journey uninterruptedly and complete this dissertation. All of these, I offer to you.

I also express my deepest gratitude to my advisors, Dr. Ravi Sandhu and Dr. Ram Krishnan, for their unparalleled support. Completing this dissertation would not have been possible without their mentoring, continual feedback, and guidance toward proper directions. I found them to be the best mentors and humans who always cared and listened to their students in every facet during this most challenging voyage. I am so proud of them. I will try to reflect on my career and life with the wisdom and experience I accumulated professionally and personally from them. Thank you, Professors, for being role models!

Moreover, I thank Dr. Yufei Huang for his help and valuable advice. He played a crucial role in shaping this dissertation from its earlier stage. I am one of the few most fortunate people who got to work with him. I would also like to extend my sincere appreciation to my doctoral dissertation committee members Dr. Palden Lama, Dr. Wei Wang, and Dr. Xiaoyin Wang, for their feedback, suggestions, time, and effort. Besides, I want to thank the faculties of the Computer Science (CS) department for their help and cooperation during my course works and Teaching Assistant (TA) roles.

I want to acknowledge James Benson and Farhan Patwa from the Institute for Cyber Security (ICS) for their unwavering support during this journey. Also, I'd like to thank and recognize the assistance I received from Suzanne Tanaka, Susan Allen, and other ICS and CS department staff members. They took care of all the administrative burdens during my studies and helped me whenever needed. I appreciate my ICS Lab colleagues, CS department peers, and other UTSA friends for valuable discussion, support, and companionship. Special thanks to my family, friends, and well-wishers for their relentless love, prayers, and encouragement. I could not write this piece without you all.

Finally, I would also like to thank the CS department and ICS for allowing me to pursue my

Ph.D. at UTSA and for supporting me throughout the years. Also, I want to thank the CREST Center For Security And Privacy Enhanced Cloud Computing (C-SPECC) through the National Science Foundation (NSF) (Grant Award #1736209) and the NSF Division of Computer and Network Systems (CNS) (Grant Award #1553696) for their support and contributions to this dissertation.

August 2022

# TOWARDS MACHINE LEARNING BASED ACCESS CONTROL

Mohammad Nur Nobi, Ph.D.
The University of Texas at San Antonio, 2022

Supervising Professors: Ravi Sandhu, Ph.D. and Ram Krishnan, Ph.D.

A common trait of current access control approaches is the challenging need to engineer abstract and intuitive access control models. This entails designing access control information in the form of roles (RBAC), attributes (ABAC), or relationships (ReBAC) as the case may be, and subsequently, designing access control rules. This framework has its benefits but has significant limitations in modern systems that are dynamic, complex, and large-scale, due to which it is not straightforward to maintain an accurate access control state in the system for a human administrator.

This dissertation proposes to exploit the power of machine learning to solve access control decision-making problems. In particular, we propose Deep Learning Based Access Control (DLBAC) by leveraging significant advances in deep learning technology as a potential solution to this problem. We envision that DLBAC could complement and, in the long-term, even replace classical access control models with a neural network that reduces the burden of attribute/policy engineering and updates. Without loss of generality, we implement a candidate DLBAC model, called $\text{DLBAC}_\alpha$, using real-world and synthetic datasets. We thoroughly investigate its performance benefits by comparing it with classical ML-based approaches and ABAC models. We demonstrate the feasibility of the DLBAC by addressing issues related to accuracy, generalization, and explainability. As DLBAC makes access decisions using a black-box neural network, we provide two approaches for understanding DLBAC decisions in human terms.

Moreover, we discuss administration practices in traditional access control system, which is managed by human administrators, making the overall administration process error-prone, tedious, and ineffective. We overcome these challenges and inefficiencies in machine learning-based access control (MLBAC) by introducing novel techniques. Our experimentation reveals that DLBAC is more efficient than classical machine learning-based systems for capturing changes in access control state across the life of a system.

We also investigate the MLBAC's adversarial attack problem, focusing on manipulating information of users and resources to gain unauthorized access. We demonstrate that it is possible to design adversarial attacks for ML models deployed for access decisions by modifying a subset of user-resource metadata. Also, we show that there is potential to reduce adversarial attacks to some extent by utilizing access control-specific constraints. Besides, to demonstrate the efficiency of MLBAC in complicated real-world settings, we implement DLBAC to decide the permission decisions of different apps on mobile devices. We show that in over 88% of cases, the DLBAC can accurately predict access permissions for different apps utilizing the characteristics of the requesting apps and the context of the device and its user. Such an outcome signifies that the DLBAC can recommend permissions and alert and stop users from granting unanticipated permissions.

Finally, we discuss challenges and future research directions related to MLBAC and DLBAC, including administration, adversarial attack, bias and fairness, verification, etc. Also, we highlight the potentiality of DLBAC to operate in tandem with traditional access control systems to monitor and reinforce traditional access control systems' decisions.

## TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

# CHAPTER 1: INTRODUCTION

## 1.1 Introduction and Motivation

Access Control Lists (ACLs) [58], Role-Based Access Control (RBAC) [118], and Attribute Based Access Control (ABAC) [64] are some of the mainstream approaches to determine users' access to resources. Commercial solutions [46] that cater to organizations employ one or more of these classical access control functionalities. While tremendous progress has been made in the realm of classical access control approaches [79], one fundamental issue has remained the same for over forty years. Skilled security administrators needed to engineer and manage accesses as only humans could develop detailed policy insights about individuals' needs within the broader organization. Clearly, this leads to all types of errors and inefficiencies [16]: there remain plenty of users with accesses that should not have those accesses (over-provisioned to ease administrative burden) and plenty of users that lack accesses that should indeed have those accesses (under-provisioned for the sake of tightened security) [49, 125]. Administrators tactfully perform a balancing act to maximize security and minimize costs. This complexity is further exacerbated with the proliferation of cloud-based applications that perform machine-to-machine access through APIs, Internet of Things (IoT), Bring Your Own Devices (BYOD), etc.

In this dissertation, we propose an automated and dynamic access control mechanism leveraging advances in deep learning technology [120] that could complement or potentially replace existing traditional access control systems. It can significantly reduce the effort and involvement of a human administrator in maintaining an access control system. This approach denoted as Deep Learning Based Access Control (DLBAC), addresses four major limitations of classical access control approaches such as RBAC and ABAC. Without loss of generality, we use the term attribute to refer to any form of traditional access control information such as roles and relationships.

**1. Attribute Engineering.** An organization typically holds a vast number of metadata about its users and resources. However, those metadata are often not meaningful *access control attributes*. As a first step, using organizational context often inferred from those metadata, administrators engineer access control specific attributes that could be used to express access control rules subsequently. This is at best an art today involving semi-formal design and requirements engineering processes [114].

**2. Policy Engineering.** After access control relevant attributes are engineered, administrators need to engineer access control rules. This is accomplished through either a manual engineering process akin to attribute engineering above or automated mining techniques that take as input a primitive form of access rules such as ACLs and generate approximate ABAC rules (or user-role assignments, in the case of RBAC) [41, 85]. We will show that, for complex situations, DLBAC generally captures the access control state of the system with more precision than other approaches which are based on policy mining and classical machine learning (ML).

**3. Policy and Attribute Update.** Policies and attributes change during the life of the system. Administrators constantly update policies, and user and resource attributes in order to maintain proper authorization in the system. This is both inefficient and insecure. It is inefficient because administrators will need to manually inspect and update policies to meet the stated change. It is insecure because the changes made are both: (a) error-prone, since the modifications may or may not satisfy the intent, and (b) inaccurate, since the modifications could have unintended changes to the access control state of the system. Another way access control states are changed in the system is through attribute update, which typically lags the necessary accesses by hours, days or months.

**4. Generalization.** Most prior approaches [24,105] that mine access control rules from simpler forms of access control states such as ACLs focus on accurately capturing the access control state as given in those ACLs. Unfortunately, that leads to poor generalization [37, 143]—that is, the ability to make better access control decisions on users and resources with attributes that were not explicitly seen during the mining process. However, this is something machine learning methods, especially deep learning, are better at. They have an innate ability to make quality predictions as long as the test sample at prediction time aligns with the training data distribution. We will show that the engineered rules typically make poor access control decisions for user-resource metadata that were not explicitly seen by the mining process.

DLBAC addresses the above issues by exploring a fundamentally different approach to how access control is designed today. As illustrated in Figure 3.1, DLBAC differs from classical approaches by making decisions based on the *metadata* of users and resources and a trained neural network. (The key distinction between the notions of *metadata* and *attributes* albeit semantic has important practical benefits, which is explained in section 3.2.1.) It accomplishes this (see Figure 3.2) by first replacing access control policies with a neural network that instead makes access control decisions. Second, the neural network is trained using raw metadata from the organization instead of laboriously engineered access control attributes. Third, in response to a suggested access control state change by an administrator, the neural network makes additional meaningful changes that are hard for that administrator to make an informed decision about. Following is a list of stimulus advantages which motivated us to use deep learning to design an access control approach:

- Access control decision-making is a classification/clustering problem where an access request is granted if its respective users and resources attributes fall under a particular cluster [76]. At present, deep neural networks are one of the best techniques to make a robust and effective classification.
- A deep neural network can learn based on raw metadata [19, 107], and there is no need for additional efforts such as feature selection, attribute engineering, role engineering, etc. Such an advantage of metadata-based learning also obviates the burdens related to attributes and roles management.
- The performance and quality of traditional approaches depends on multiple factors. Other than access control state of an existing system, the performance also depends on how well one can design attributes and roles, how accurately one can assign attributes or roles to respective users, and so on. However, in the case of a deep neural network-based system, the system's performance solely depends on how effectively one can train the network based on existing access control state. Also, there are multiple metrics to quantify the quality and

effectiveness of such systems.

Moreover, access control systems are not static—changes in access control state are inevitable. A user may be granted new permissions, or some of her current permissions could get revoked. This problem is referred to as *access control administration* in the access control domain [116]. Administration challenges could vary from model to model, but the problem's importance remains unchanged. In the case of RBAC, administration activities include assigning/removing permission to/from a role, creating a new role, and managing role hierarchy [116]. For ABAC, administration activities include updating user/resource attributes and policy modification [122]. In such traditional approaches, the changes are accomplished by modifying existing configurations such as written access control policies and attribute and role assignments. However, for a machine learning based access control (MLBAC) system, there is no notion of a human-readable written policy to update. If an access control state change is to be made, the existing ML model must be modified. Such a modification is complicated as, in most cases, an ML model is a highly complex function, a tree, or even a black box that a human user can not directly access and modify. Therefore, it needs to investigate the administration problem if an MLBAC is deployed for access control decisions.

Also, neural networks are prevalent for obtaining a generalized and accurate system due to their ability to capture features from complex input [30, 77, 109]. Such quality of the neural networks could pose some security concerns as they are highly sensitive to minor changes in the input—that is, a slight manipulation or introduction of additional information to the input may result in an unintended output [14]. In ML, this issue is known as an *adversarial attack*. For instance, in access control, an attacker could manipulate the user/resource metadata to gain access to a resource forcibly. As a result, it requires a thorough investigation of adversarial attacks in the context of MLBAC.

## 1.2 Problem and Thesis Statement

### 1.2.1 Problem Statement

The policy-based classical access control systems are limited in a complex and dynamic environment where the access control state may vary with subtle changes in the user and resource attributes. To tackle such a dynamicity to some degree, system administrators rely on (1) creating complex rules with many conditions and constraints or roles with a large number of permissions and (2) making more fine-grained rules or roles leading to rule/ role explosion. While the former makes access control maintenance very difficult for a human administrator, the latter granulates the access control state, which is less generalized and maintainable.

Therefore, it is essential to develop an access control system that can accommodate complex and dynamic access control state and is generalized enough to make accurate decisions for unseen access control requests.

### 1.2.2 Thesis Statement

*A deep neural network can precisely learn the access control state of a large-scale, complex, and dynamic system, generalize enough to make accurate decisions for unseen access control requests and ease access control administration by employing processes with minimal human involvement..*

## 1.3 Summary of Contributions

We make the following contributions to the field of access control:

- This dissertation performs a comprehensive review of existing access control literature that uses machine learning. We also propose a novel taxonomy of machine learning in access control, assemble publicly available real-world datasets for studying and solving access control issues, and highlight research at each stage as the domain evolved.

- We propose Deep Learning Based Access Control (DLBAC) as a potential form of machine learning based access control (MLBAC) system. We show that the DLBAC, in the best case, could replace and operate as a complement to classical access control approaches in the normal case. We validate the efficacy of DLBAC by implementing a prototype, named DLBAC$_\alpha$, using real-world and synthetic datasets that outperform classical policy mining and machine learning techniques.

- We address previously highlighted concerns on the explainability of the black-box nature of neural network-based systems for access control [30]. We propose two unique methods to understand DLBAC decisions in human terms and confirm that the rationale behind a DLBAC decision can be understood largely (albeit not with 100% accuracy).

- We explore MLBAC's administration problem, focusing on capturing changes in the access control state. We propose novel methods for efficient DLBAC administration. While compared with the classical ML-based system's administration, DLBAC performs better in adjusting changes in the access control state.

- We investigate vulnerabilities of the MLBAC system in an adversarial setting in the context of access control. Also, we show that there is potential to reduce adversarial attacks to some extent by utilizing access control-specific constraints.

- We implement DLBAC to decide the permission decisions of different applications on mobile devices. We show that DLBAC can recommend permission requests on Android devices with good accuracy. At the very least, one could utilize such a recommendation to warn and prevent users from granting unanticipated permissions.

## 1.4 Dissertation Organization

In this chapter, we discuss our motivations for this work. Also, we identify the major problems in traditional access control systems as a whole and summarize our key contributions to solving those problems. Chapter 2 thoroughly reviews existing access control literature that uses machine learning and proposes a taxonomy of machine learning in access control. This chapter also briefly discusses different methods and summarizes their application and context.

Chapter 3 presents DLBAC as a potential form of MLBAC and shows how DLBAC fundamentally differs from traditional access control approaches. This chapter also illustrates the synthetic data generation method for a prototype of the DLBAC model named DLBAC$_\alpha$. The same chapter explains the implementation of DLBAC$_\alpha$ with its performance evaluation. This chapter also proposes two approaches for understanding DLBAC$_\alpha$ decisions in human terms.

Chapter 4 discusses administration problems and requirements in MLBAC with formalizing MLBAC administration architecture. This chapter also implements two MLBAC administration prototypes. It also discusses various issues that could arise while performing MLBAC administration and demonstrates how to overcome such challenges. Chapter 5 investigates the adversarial attacks in the MLBAC system in an adversarial environment and proposes a novel technique to tackle these vulnerabilities.

To demonstrate the efficiency of MLBAC in complicated real-world settings, we implement a DLBAC for the permission decisions of different apps on mobile devices in Chapter 6. Besides, this chapter identifies and discusses potential applications where a DLBAC can work in tandem with traditional access control systems. The same chapter also discusses the feasibility of DLBAC in IoT systems. Finally, Chapter 7 discusses some future research directions and concludes the dissertation.

# CHAPTER 2: BACKGROUND AND LITERATURE REVIEW

This chapter explores the current practices of machine learning (ML) in access control and determines the future perspectives of this fast-growing domain. The outcome of this research has been submitted for publication to the following journal:

- Mohammad Nur Nobi, Maanak Gupta, Lopamudra Praharaj, Mahmoud Abdelsalam, Ram Krishnan, and Ravi Sandhu. "Machine Learning in Access Control: A Taxonomy and Survey." ACM Computing Surveys (CSUR).

## 2.1 Introduction

The effect of ML in access control is found to be positive and has huge potential to achieve even more. Since the use of ML for access control purposes is rapidly emerging, several challenges are encountered in the current ML-oriented access control research practice. We observe that researchers applied ML methods on a case-by-case basis [6,27,37,50,78,138], thereby, there is no common strategy for using ML in the access control domain. The lack of an exhaustive overview of the application of ML in access control makes it even harder to gain in-depth insights into it and plan accordingly. In addition, limited research efforts to determine the proper ML method for a distinct access control problem make it challenging to select the best model for a novel problem.

Besides, there are several other limitations, such as a lack of quality datasets from the real-world organization [45,61] or the anonymous publicly available datasets may not contain relevant access control information capable of expressing a complete access control state of a system [102]. Putting all these into perspective, it is essential to have a holistic view of how researchers use machine learning for access control. Also, such a cohesive picture will help learn under-developed areas and determine future research in the domain.

In this chapter, we perform a detailed review and summarize existing literature that uses ML to solve different access control problems, including attribute and role extraction, policy mining, access control decision making, and policy verification. We also propose a novel taxonomy of machine learning in access control and highlight research at each stage as the domain evolved chronologically. Besides, we summarize the publicly available real-world datasets used for machine learning-based access control research. In the end, we highlight open challenges and limitations faced by the research community and provide future research directions to thrive in this critical security domain.

## 2.2 Overview of Access Control

### 2.2.1 Access Control Approaches

As the cornerstone of any information system's security, access control restricts the authorization of subjects/users to perform operations on system's resources/objects. Different abstraction, models and mechanisms have been proposed to govern the access and operations on resources to better represent the access control state of a system at various granularity levels with better maintainability. We briefly discuss some mainstream access control approaches and models in this section.

**Access Control List (ACL)**

The ACL is the low-level representation of the access control state of a system. An ACL informs a system of a user's access privileges to system objects/resources (e.g., a file or a directory) [119]. Each object has a security property that connects it to its access control list. The list has an entry for every user with access rights to the system. However, as each user and resource is managed separately based on an identifier, ACLs make it very difficult to administer.

**Discretionary Access Control (DAC)**

It is an access control model where the *Owner* or *Security Administrator* of a system sets policies defining who can access the system resources [119]. DAC is efficiently implemented using ACL or *capabilities* [132]. The security administrator defines a resource profile for each object and updates the access control list for the profile. In DAC, it is difficult to enforce the principle of least privilege and separation of duties [44].

**Mandatory Access Control (MAC)**

In the MAC model, the administrator makes the policy decision, which is based on the security labels of subjects (clearance) and objects (classification) [117]. The user does not control, define or change access rights. A common example of MAC is in military security, where the data owner does not decide on who has a top-secret clearance, or change the classification from top-secret to secret [65]. In MAC, configuration and maintenance of the system is a challenging task for the administrator, along with covert channel attacks.

**Role Based Access Control (RBAC)**

In RBAC, access permissions depend on users' roles in the system [118]. An administrator designs the role defining what permissions a role should have and which users will be assigned what roles. For example, some users may be assigned to a role that allows them to read and write a file, whereas others may be assigned a different role, which constrains them to read the file only. A user can be assigned many roles, and at the same time, many users may share one single role. Roles in RBAC offer easy permissions administration, as role clusters the abilities a user can hold. RBAC also offers static and dynamic separation of duties, with the ability to define other constraints such as mutually exclusive roles, permissions, and cardinality constraints. However, RBAC is vulnerable to role explosion problems, which can lead to leakage of permissions as the number of users and roles in an organization expands.

**Attribute Based Access Control (ABAC)**

In ABAC, a user's access to a resource is determined based on the attributes of the user, attributes of the resource, environmental conditions, and a set of pre-defined policies [66]. An ABAC policy defines the combinations of the user, resource, and/or environmental attributes, and eventually, these attributes are required to allow the user to operate on a resource. For example, to restrict a department's resource to its 'sales_manager' only, the ABAC policy could limit the rule so that users with the 'job_role' attribute have the value 'sales_manager' can access the respective resource. As the access policy is designed with respect to rules, ABAC is one of the most generalized, scalable, and reliable access control models [122]. However, there are challenges while implementing ABAC as it involves multiple laborious sub-processes such as *attribute engineering*, *attributes assignment*, *policy engineering*, etc. Therefore, a skilled human administrator might need to be involved in designing ABAC rules and related attributes to satisfy overall security policy requirements.

#### Relationship Based Access Control (ReBAC)

Relationship-based access control defines access decisions based on the *relationships* between subjects [27, 48] or objects. The most well-known examples of relationship-based access control are social networks. On Facebook, for example, the user gives view access to the photos or videos to friends or friends of friends. But friends of those friends cannot view the photos. Thus, ReBAC allows access when the user has a certain relationship with other entities in the system. ReBAC offers more than other access control models because it grants access based on multi-relationship between entities and takes decisions for certain entities, not entity types. For example, the user can access a specific photo from the directory, not the total directory.

### 2.2.2 Access Control Concepts

#### Policy Mining

Policy Mining is the automation process of access control policy extraction from an existing access control state. Migration to a new model manually can be an error-prone and time-consuming task. To reduce the effort and cost of this task, extracting policies from the given access control information can be partly or completely automated [72]. For example, if an organization had already an access control model implementation such as ACLs, and wants to migrate to more flexible, fine-grained access control model such as RBAC, ABAC, or ReBAC. In this case, an ABAC or ReBAC policy mining approach can be applied to obtain desired policies in automated and efficient manner [143]. In general, the output of ABAC or ReBAC mining is a set of rules [26], whereas, the RBAC policy mining approaches produce permission to roles (PA) and user to roles

(UA) assignments [108, 147]. There are several approaches that use advancements in machine learning to automate the ABAC [6, 8, 37, 74, 78], RBAC [50, 60, 147], and ReBAC [25–27] policy mining process.

### Policy Verification and Testing

Access control policy verification confirms no flaws within the policy that leaks or blocks access privileges. As a software test, access control policy verification depends on model proof, data structure, system simulation, and test oracle to validate that the policy logic functions are working properly. However, these methods have capability and performance concerns related to inaccuracy and complexity limited by applied technologies. For instance, model proof, test oracle, and data structure methods assume that the policy under verification is flawless unless the policy model cannot hold for test cases. Thus, the challenge of the method is to compose test cases that can systematically learn all faults.

Moreover, a system simulation method needs to convert the policy to a simulated system. But, if the policy logic is complex or the number of policy rules is large, the translation between the system may be difficult or impractical [65]. To overcome these challenges machine learning approach is used for policy verification which does not require comprehensive test cases, oracle, or system translation. Rather, it checks the logic of the policy rule directly, making it more efficient and feasible compared to the traditional method [61, 65].

### Policy Administration and Monitoring

Designing the policies is not easy and requires substantial administrative effort to modify the policy to accommodate changes. Multiple ML-assisted methods support additional adjustments in the existing access control policy and help detect improper behavior in the access control system. ML-based techniques can help system administrators to reduce the cost of generating the policies and adjust them dynamically [4,56]. Besides, to accommodate changes in the system, the system admin may introduce errors and misconfiguration to the system due to the dependencies on the user, data,

**Table 2.1**: Publicly Available Real-world Datasets Used in Access Control Researches

| SL. | Name | Publish Year | Reference | Type | Description | Application |
|---|---|---|---|---|---|---|
| 2.1.1 | IBM-CM | 2004 | IBM [1] | Access Policies | Natural language access control policy | [7], [8] |
| 2.1.2 | University-Data | 2005 | Fisler et al. [47] | Access Policy | Central grades repository system for a university | [95] |
| 2.1.3 | Wikipedia | 2009 | Urdaneta et al. [134] | Access Logs | Access request traces from Wikipedia | [138] |
| 2.1.4 | AmazonUCI | 2011 | UCI Repository [11] | Access Logs | Access data of Amazon employees | [30], [37], [74], [76] |
| 2.1.5 | iTrust | 2012 | Meneely et al. [100] | Access Policies | Natural language access control policy | [7], [8] |
| 2.1.6 | CyberChair | 2012 | Stadt et al. [136] | Access Policies | Natural language access control policy | [7], [8] |
| 2.1.7 | Collected-ACP | 2012 | Xiao et al. [139] | Access Policies | Natural language access control policy collected from multiple sources | [7], [8] |
| 2.1.8 | Amazon-Kaggle | 2013 | Kaggle [10] | Access Logs | Two years historical access data of Amazon employees (12000 users and 7000 resources) | [30], [37], [77], [76], [93] |
| 2.1.9 | eDocument | 2014 | Decat et al. [42] | Access Policy | e-document case study | [25], [26], [27] |
| 2.1.10 | Workforce | 2014 | Decat et al. [43] | Access Policy | Workforce management case study | [25], [26], [27] |
| 2.1.11 | SCADA-Intrusion | 2015 | Turnipseed et al. [133] | SCADA Data | SCADA dataset for intrusion detection system | [147] |
| 2.1.12 | Dalpiaz | 2018 | Dalpiaz et al. [39, 40] | User Stories | Over 1600 user stories from 21 web applications | [60] |
| 2.1.13 | Incident | 2018 | Amaral et al. [9] | Event Logs | Event log from an incident management process | [30] |

functionality, and domain [12, 138]. Despite efforts on verification and testing [65] to avoid access control misconfiguration, it is still difficult to avoid in the real-world system. An automated real-time monitoring system could help to observe the changes in access control behavior better and act accordingly [95, 138].

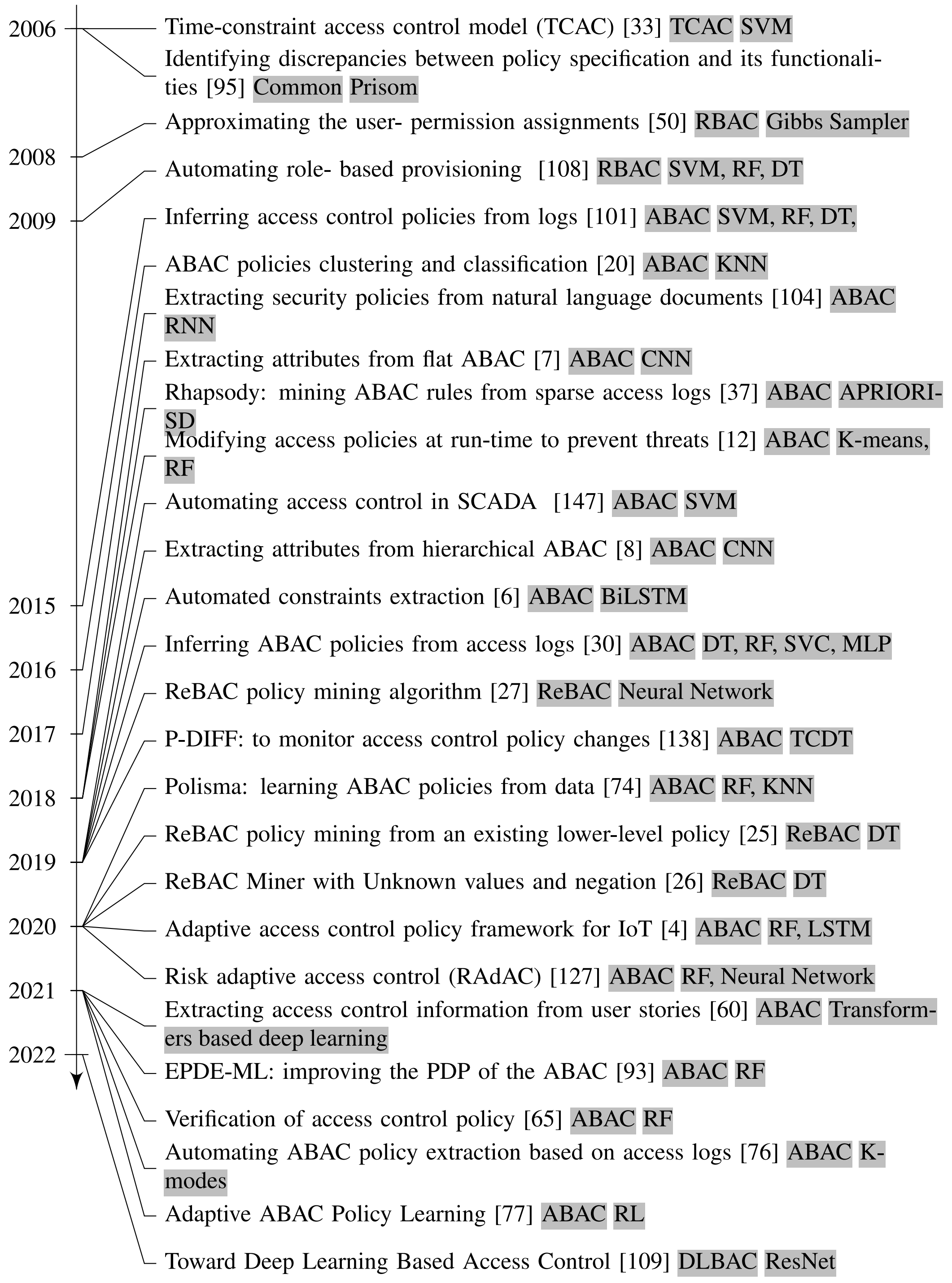
2006
2008
2009
2015
2016
2017
2018
2019
2020
2021
2022
Time-constraint access control model (TCAC) [33] TCAC SVM
Identifying discrepancies between policy specification and its functionalities [95] Common Prisom
Approximating the user- permission assignments [50] RBAC Gibbs Sampler
Automating role- based provisioning [108] RBAC SVM, RF, DT
Inferring access control policies from logs [101] ABAC SVM, RF, DT,
ABAC policies clustering and classification [20] ABAC KNN
Extracting security policies from natural language documents [104] ABAC RNN
Extracting attributes from flat ABAC [7] ABAC CNN
Rhapsody: mining ABAC rules from sparse access logs [37] ABAC APRIORI-SD
Modifying access policies at run-time to prevent threats [12] ABAC K-means, RF
Automating access control in SCADA [147] ABAC SVM
Extracting attributes from hierarchical ABAC [8] ABAC CNN
Automated constraints extraction [6] ABAC BiLSTM
Inferring ABAC policies from access logs [30] ABAC DT, RF, SVC, MLP
ReBAC policy mining algorithm [27] ReBAC Neural Network
P-DIFF: to monitor access control policy changes [138] ABAC TCDT
Polisma: learning ABAC policies from data [74] ABAC RF, KNN
ReBAC policy mining from an existing lower-level policy [25] ReBAC DT
ReBAC Miner with Unknown values and negation [26] ReBAC DT
Adaptive access control policy framework for IoT [4] ABAC RF, LSTM
Risk adaptive access control (RAdAC) [127] ABAC RF, Neural Network
Extracting access control information from user stories [60] ABAC Transformers based deep learning
EPDE-ML: improving the PDP of the ABAC [93] ABAC RF
Verification of access control policy [65] ABAC RF
Automating ABAC policy extraction based on access logs [76] ABAC K-modes
Adaptive ABAC Policy Learning [77] ABAC RL
Toward Deep Learning Based Access Control [109] DLBAC ResNet

**Figure 2.1**: A Timeline of Seminal Works Towards ML in Access Control. In each work, the first grey highlight indicates the access control model ('Common' implies the method is applicable for any access control model), and the second highlight denotes ML algorithms applied in the corresponding method.

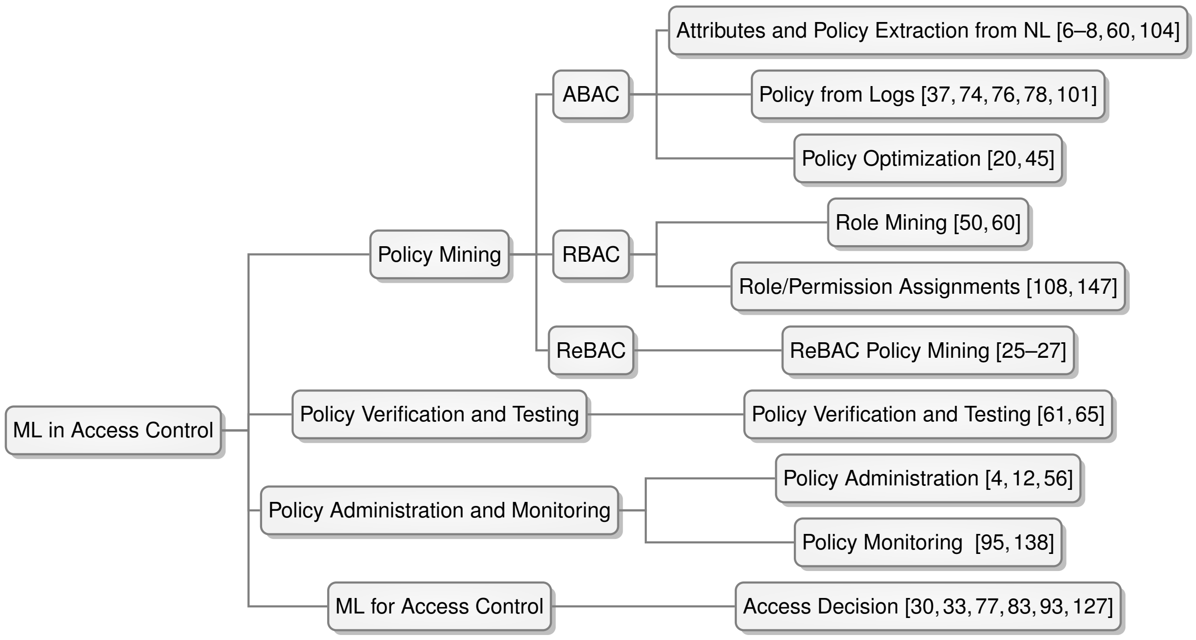
ML in Access Control
Policy Mining
ABAC
Attributes and Policy Extraction from NL [6–8, 60, 104]
Policy from Logs [37, 74, 76, 78, 101]
Policy Optimization [20, 45]
RBAC
Role Mining [50, 60]
Role/Permission Assignments [108, 147]
ReBAC
ReBAC Policy Mining [25–27]
Policy Verification and Testing
Policy Verification and Testing [61, 65]
Policy Administration and Monitoring
Policy Administration [4, 12, 56]
Policy Monitoring [95, 138]
ML for Access Control
Access Decision [30, 33, 77, 83, 93, 127]

**Figure 2.2**: A Taxonomy of Machine Learning in Access Control Domain. NL indicates Natural Language.

## 2.3 Machine Learning in Access Control

### 2.3.1 Policy Mining

Due to the flexible policies and fewer management burdens, high-level access control models such as ABAC and ReBAC are adopted to support dynamic and complex security policies. However, adopting such policies from an existing lower-level policy like ACLs is challenging. Generally, the policy mining techniques take attributes of users/resources in the system and the current access control state of the system as input. For ABAC and ReBAC, the algorithm output a set of rules (policy) that grants the same permissions [27, 74]. In contrast, RBAC mining algorithms output permission to roles (PA) and user to roles (UA) assignments [108, 147]. Table 2.2 summarizes access control policy mining approaches using machine learning.

#### Attribute Based Access Control (ABAC)

**Attributes and Policy Extraction from Natural Language.** Natural language policies, being the preferred expression of policy [7], need to be transformed into a machine-readable form. Several researchers attempted to process such policies using ML to extract access control-related information, including identifying policy sentences, triples of subject-object-action, etc.

Narouei et al. [104] introduce a top-down policy engineering framework for ABAC that automatically extracts security policies from natural language (NL) documents using deep recurrent neural networks (RNN). Alohaly et al. [7] propose a deep learning-based automated approach for extracting ABAC attributes from NL policy. The authors develop a framework to automate the attribute extraction and related information leveraging natural language processing (NLP), relation extraction (RE), and CNN.

The primary goal of the Alohaly et al. [7] is to detect values of attributes (attribute extraction) in ABAC policies with no hierarchy among subject or object elements (*flat ABAC*). From a practical perspective, extraction of authorization attributes of *hierarchical ABAC* system from natural language artifacts is more needed than the flat counter-part. The authors in [8] extend the framework proposed in [7], built upon recent advancements in NLP and ML techniques, with automatic attributes extraction ability from NL hierarchical ABAC policies.

In ABAC, the attributes often need to satisfy some constraints, which is critical for complying with organizational security policy. It requires extraordinary skills and efforts to define the constraints formally. The process is tedious and error-prone as security architects have to analyze several documents to develop them manually. The Alohaly et al. [6] proposed a reliable and automated constraints extraction process exploiting tools in NLP. This automation also enables trace-ability formal constraints expressions and related policies that will also help to avoid errors while assigning unauthorized attributes.

The user stories written by software developers better represent the actual code than high-level product documentation. Such stories can contain access control-related information, and extracting that information can be used to construct access control documentation. Also, stakeholders can use this information for access control engineering, development, and review. Authors in [60] developed an automated process using ML to extract access control information from a set of user stories that present the behavior of the software product in question. The proposed work takes a collection of user stories as input to a transformers-based deep learning model [**?**] with three components: access control classification, named entity recognition, and access type classification. First, it classifies whether a user story contains access control information or not. Then, it identifies the actors, data objects, and operations in the user story as part of a named entity recognition task.

**Policy from Access Logs.** Work by Mocanu et al. [101] proposed a deep learning-based approach to infer policies from logs, which also supports negative authorization (i.e., a subject cannot access a resource). The proposed approach has two phases. The first phase generalizes the knowledge from logs providing a set of candidates rules in a binary vector format. In the second phase, this set of candidate rules is transformed to the format acceptable by Xu-Stoller [143] and compared. However, such kinds of methods [101, 143] are limited where the logs have only a tiny subset of all possible access requests. Evidently, the access logs in real-world systems only contain about 10% of all possible access requests [37]. Over time, the developed rules also become highly complex due to its adoptions to different changes in the system. Hence, the shorter and less-convoluted rules are better from a maintainability perspective. Cotrini et al. [37] proposed an approach, known as Rhapsody, for mining ABAC rules from sparse access logs identifying patterns among the authorized requests. Rhapsody also solves various other issues of the existing state-of-the-art, including the rules size and over-permissiveness nature. The Rhapsody builds upon APRIORI-SD [80], which the authors modify to overcome its weaknesses of mining overly permissive or unnecessarily large rules.

Cotrini et al. also introduce a novel validation method named *universal cross-validation* that evaluates mined rules with requests not present in the dataset. Experiments show that the proposed evaluation approach validates policies with higher F1 scores (more accurate in deciding requests outside the log) than the standard cross-validation technique.

Jabal et al. [74] propose a novel framework for learning ABAC policies, named Polisma, combining data mining, statistical, and ML techniques. The methodology of the Polisma approach works mainly in four different stages. Polisma applies Random Forest (RF) and KNN as the machine learning classifiers on requests not covered by the set of policies learned in the first three stages and uses the classification result to label these data and generate additional rules. The approach is evaluated empirically using both the real-world [11] and synthetic datasets. Experimental results show that Polisma can develop ABAC policies that accurately control access requests.

Many large-scale businesses need to grant authorizations to their users that are *distributed* across heterogeneous computing environments. In such a case, the manual development of a single access control policy framework for an entire organization is cumbersome and error-prone. Karimi et al. [76] proposed an automating ABAC policy extraction based on access logs following their other work on an unsupervised learning-based approach for ABAC policy mining [78]. The proposed method extracts policy rules containing positive and negative attributes and relationship filters.

**Policy Optimization.** Benkaouz et al. [20] presents an approach for ABAC policies clustering and classification. The proposed approach uses KNN algorithms that help to reduce dimensionality and achieve higher flexibility for ABAC policies in high-scale systems. A small $k$ implies fine-grained ABAC model, and a bigger $k$ implies a coarse-grained ABAC model. This is a work in progress, and several key questions remain unanswered, such as the default value of $k$, the most suitable KNN algorithms for policies clustering, and also if the same approach has any uses in different kinds of applications. El Hadj et al. [45] propose ABAC-PC (ABAC Policy Clustering) that groups the policy rules according to the decision effects, such as to permit or deny rules. The method also creates cluster rules based on similarity scores and produces the minimum set of rules representing each cluster. The proposed work is an extension of the Benkaouz et al. [20] method.

### Role Based Access Control (RBAC)

The work in [50] focuses on bottom-up approaches for RBAC role mining where the goal is to approximate the user-permission assignments by finding a minimal set of roles, user-role and role-permission assignments. The authors use Gibbs sampler [106] which is a Markov chain Monte

Carlo algorithm for their proposed Disjoint Decomposition Model.

The authors in [108] propose a machine learning-based automated role maintenance system that provisions existing roles with entitlements from newly deployed applications and provisions new users with existing roles. The proposed technique utilizes the attributes of entitlements to simplify the management of entitlements assignments to roles. Lu Zhou et al. [147] propose two machine learning-based approaches to automate the role assignment process in the context of the real-time Supervisory Control and Data Acquisition (SCADA) system. The authors first demonstrate how SVM can be applied by developing a context-aware RBAC with automated role engineering. They use different static (job function, job position, etc.) and dynamic attributes (time, location, etc.) of the SCADA system's users and devices as input to the SVM and obtain user-role or permission-role assignments as the output.

**Relationship Based Access Control (ReBAC)**

Like ABAC policy mining approaches, ReBAC policy mining algorithms can also potentially reduce the effort to obtain a high-level policy from lower-level access control data. The author in [27] propose an efficient and scalable ReBAC policy mining algorithm which is an extension over the existing evolutionary algorithm [29] for the same problem. This enhancement simplifies the current approach, facilitating the easier adoption of new policy language features. Due to the flexibility and scalability of high-dimensional data and large datasets, the authors choose a neural network over other classification methods for the feature selection algorithm. The algorithm maps different *feature vectors* with Boolean values to indicate the authorization for users to act an action on resources. Each *feature* in the vector is defined as 'a subject atomic condition, resource atomic condition, or atomic constraint satisfying the user-specified limits on lengths of paths in conditions and constraints'.

The authors in [25] proposed DTRM (Decision Tree ReBAC Miner) and DTRM$^{-}$, with the ability to mine policies in any ReBAC language. The decision tree-based DTRM algorithm mines policies in the recent version of Object-oriented Relationship-based Access-control Language

(ORAL) that supports two set comparison operators on top of existing operators in the earlier version. The DTRM$^-$ algorithm is an extension of DTRM with the support of negative conditions and negative constraints. Similar to [27], the decision tree is trained to learn the classification of feature vectors, which are also mapped with Boolean values indicating the authorization for users to act on resources.

In most real-world data, information about permissions can be incomplete, or some attribute values can be missing (or unknown). Authors in [28, 37, 73] solved different variants of the ABAC and ReBAC policy mining problem considering *incomplete permissions information*. However, all these works assume the attribute (and relationship in the case of ReBAC) information is complete (or known). The authors in [26] proposed decision tree-based algorithms, Decision-Tree ReBAC Miner with Unknown values and negation (DTRMU$^-$ and DTRMU), for mining ABAC and ReBAC policies from ACLs. The first of its kind, the algorithm can deal with 'incomplete' information about entities, where the values of some attributes of some entities are unknown. To handle the 'unknown' values, the authors proposed to reduce the core part of the ReBAC policy mining problem to the general problem of Kleene's three-valued logic learning formula. The authors assigned another truth value, 'U,' to conditions and constraints with 'unknown' values along with true (T) and false (F) binary logic values.

To deal with a concise three-valued logic formula from a set of labeled feature vectors involving unknowns, the authors develop an algorithm based on multi-way decision trees as opposed to a binary decision tree. Identical to [25, 27], the decision tree learns based on a given set of *feature vectors* labeled with a permit or deny. The definition of feature is also same as [27] and [25].

### 2.3.2 Policy Verification and Testing

Traditional policy verification methods are infeasible for many systems as they are error-prone and time-consuming processes. These methods perform static or dynamic analysis on the access control system to determine the policy behavior. Such analysis techniques can not specify roles or permissions-specific code elements, find relations among these codes and associated policy

**Table 2.2**: Summarizing Machine Learning Based Policy Mining. The dataset type 'RW', 'RWA', and 'Syn' indicates Real-World, Real-World Augmented, and Synthetic dataset, respectively. We link to Table 2.1 if the respective RW/RWA dataset is public. Also, we follow the same convention for other summary tables.

| Reference | Problem Considered | Access Control Model | ML Approach | Dataset Type |
|---|---|---|---|---|
| Frank et al. 2008 [50] | Probabilistic bottom-up approaches for RBAC role mining | RBAC | Gibbs sampler & Disjoint Decomposition | Syn & RW |
| Ni et al. 2009 [108] | Mapping between roles and new privileges | RBAC | SVM (and DT, RF, etc.) | Syn & RW |
| Mocanu et al. 2015 [101] | Policy inference from logs | ABAC | RBM | Syn |
| Benkaouz et al. 2016 [20] | Clustering of policies | ABAC | KNN | Not Used |
| Narouei et al. 2017 [104] | Policy extraction from natural language documents | ABAC | Recurrent Neural Network | Syn |
| El Hadj et al. 2017 [45] | Clustering of policies | ABAC | KNN | Syn |
| Alohaly et al. 2018 [7] | ABAC attribute extraction from natural language | ABAC | CNN | RWA (Table 2.1.1, 2.1.5, 2.1.6, 2.1.7) |
| Karimi et al. 2018 [78] | Policy extraction | ABAC | K-modes | Syn |
| Cotrini et al. 2018 [37] | Policy mining | ABAC | APRIORI-SD [80] | Syn & RW (Table 2.1.4, 2.1.8) |
| Alohaly et al. 2019 [8] | Attribute extraction from natural language | ABAC | CNN | RWA (Table 2.1.1, 2.1.5, 2.1.6, 2.1.7) |
| Alohaly et al. 2019 [6] | ABAC constraints extraction from natural language | ABAC | BiLSTM | RWA |
| Zhou et al. 2019 [147] | Role and permission assignments | RBAC | SVM & Adaboost | RWA (Table 2.1.11) |
| Bui et al. 2019 [27] | Policy mining | ReBAC | Neural Network | Syn & RW (Table 2.1.9, 2.1.10) |
| Bui et al. 2020 [25] | Policy mining | ReBAC | Decision Tree | Syn & RW (Table 2.1.9, 2.1.10) |
| Bui et al. 2020 [26] | Policy mining | ABAC, ReBAC | Decision Tree | Syn & RW (Table 2.1.9, 2.1.10) |
| Jabal et al. 2020 [74] | Learn ABAC policies from logs | ABAC | RF, KNN | Syn & RW (Table 2.1.4) |
| Karimi et al. 2021 [76] | Policy extraction based on access logs | ABAC | K-modes | Syn & RW (Table 2.1.4, 2.1.8) |
| Heaps et al. 2021 [60] | Extracting access control policy from user stories | RBAC and ABAC | Transformers, CNN, SVM | RW (Table 2.1.12) |

**Table 2.3**: Summarizing Machine Learning Based Policy Verification. 'Common': any access control model.

| Reference | Problem Considered | Access Control Model | ML Approach | Dataset Type |
|---|---|---|---|---|
| Heaps et al. 2019 [61] | Word embedding technique for developing automated policy verification tools | RBAC | Neural Network | Syn |
| Vincent C. Hu 2021 [65] | Verifying access control policy | Common | RF | Syn |

elements, or tackle the case when a mapping is required for a new code and policy element. The authors in [61] discuss these issues and pretend a potentiality of developing a more robust and efficient system by leveraging recent advancements in deep learning. The authors propose training a deep learning model based on the code and policy elements links. Access control policies are verified based on model proof, data structure, system simulation, and test oracle to ensure the policy works desirably. By default, it is assumed that the policy under verification is accurate unless the policy failed to hold for test cases. However, this comprehensive test case generation is challenging and somewhat impractical. In a NIST Internal Report (IR), Vincent C. Hu [65] proposes a technique exploiting machine learning algorithm to make the entire verification process more efficient and straightforward. The proposed method, which is feasible and efficient, can check the policy logic directly compared to traditional methods. Table 2.3 outlines these works.

### 2.3.3 Policy Administration and Monitoring

A regular access control policy update a.k.a policy administration is required to accommodate intermediate access changes. The task is very challenging for system admins and error-prone (e.g., over-granting access privilege) due to the lack of proper tools [138]. Manual policy updates are a laborious and error-prone task and the system becomes vulnerable to different cyber threats [12]. Also, failing to detect such threats (or misconfigurations) in the early stage may cause severe security incidents. A body of methods have been proposed to automate the policy administration and policy monitoring processes. Table 2.4 lists related methods. We briefly discuss them below.

### Policy Administration

Authors in [12] present an ML-based approach, named ML-AC, that can update policies at run-time automatically and prevent such threats. The ML-AC monitor and learn the *access behavioral features* (e.g., frequency of access, amount of data, location, etc.) of a user and adjust existing access control rules based on learned *contextual knowledge*. The authors apply this method by adding a novel Contextual Behaviour Learning component in the Policy Administration Point (PAP) of the access control system. This new component builds user profiles based on the monitored access pattern and adjusts access control policies by utilizing those profiles.

The authors in [4] propose an adaptive access control policy framework for IoT deployments. The framework dynamically refines the access policies based on the access behaviors of the IoT devices. In parallel to the traditional ABAC authorization server, the authors propose incorporating a policy management module that implements the access policy adaptation functionalities, including the access behavior classifier and the policy refinement components.

Gumma et al. [56] propose PAMMELA which is an ML-based ABAC policy administration method. The PAMMELA creates new rules for the proposed changes and extends existing policy by adjusting the newly developed rules with the current policy. PAMMELA can also develop new policies for a system by learning existing policy rules in a similar system. PAMMELA is a supervised ML-based approach that mainly works in two phases. PAMMELA trains an ML classifier on ABAC policy rules in the first phase. In the second phase, PAMMELA provides a set of access requests to the trained classifier and creates a set of rules based on the classifier's access decision for respective requests.

### Policy Monitoring

Due to the lack of proper tools, standardized policy specifications such as XACML and their implementation can be error-prone. It demands rigorous verification and validation to ensure the implemented policy complies with the desired one. Martin et al. [95] discover that the ML algorithms could summarize the basic properties of such a policy and help to identify specific bug-exposing

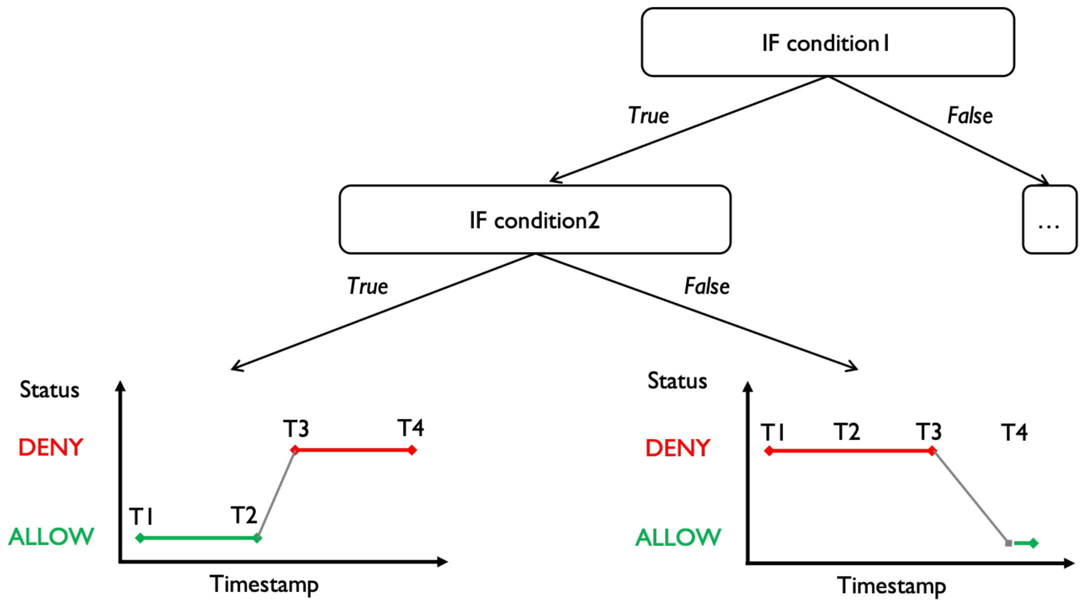

**Figure 2.3**: Time-Changing Decision Tree (TCDT).

requests. Especially, their proposed method can efficiently identify the discrepancies between the policy specification and its intended functionalities.

Authors in [138] develop a tool named P-DIFF to backing system admins to monitor access control policy changes. P-DIFF also supports investigating any malicious access by backtracking related changes that help system admins identify the reason behind unwanted access. The authors adopt a decision tree representation to convert heterogeneous access control behaviors to a standard rule-based format. However, a regular decision tree is limited in handling time-series information to consider access concerning time. The authors extend the decision tree algorithm to counter the challenge and propose a novel Time-Changing Decision Tree (TCDT). Unlike single binary value outcome (allow or deny), each rule of a TCDT is expressed as a time series as depicted in Figure 2.3. The TCDT learning algorithm is designed to infer the tree by considering access logs as a sequence of access events ordered by access time. Then, the TCDT allows modeling access control behavior at any given time and monitoring changes in the access control rules.

**Table 2.4**: Summarizing Machine Learning Based Policy Administration and Monitoring. 'Common': any access control model.

| Reference | Problem Considered | Access Control Model | ML Approach | Dataset Type |
|---|---|---|---|---|
| Martin et al. 2006 [95] | Identifying bug-exposing requests | Common | Prism [**?**] | RW (Table 2.1.2) |
| Argento et al. 2018 [12] | Improves the PAP of the ABAC model | ABAC (and Common) | RF and K-means Clustering | Syn |
| Xiang et al. 2019 [138] | Access control validation and forensics | Common | Time-Changing Decision Tree (TCDT) | RW (Table 2.1.3) |
| Ashraf et al. 2020 [4] | Refines policies based on access behavior | ABAC | RF and RNN | RW |
| Gumma et al. 2021 [56] | ABAC policy administration | ABAC | Neural Network, DT, RF, etc. | Syn |

### 2.3.4 ML for Access Control Decision Making

Contemporary researches also manifest the advantages of using an ML model for more accurate access control decision-making [30,33,77,93,109,127]. These systems decide accesses based on a trained ML model instead of a written access control policy. Generally, these models make access control decisions (grant or deny) using user and resource metadata and attributes. Metadata and attributes are the user/resource features that an ML model learns for subsequent access decisions. We briefly discuss these approaches below and summarize them in Table 2.5.

Chang et al. [33] proposed a novel access control system, called time-constraint access control, that can be applied to access policies constrained with time. The authors apply SVM to the proposed scheme and divide the processes into three phases: (1) the input pattern transforming, (2) the training phase for SVMs, and (3) the authority decision phase. As part of training data, the system administrator determines a login time and a password for each user. The trained SVMs can classify the users into their groups and give them corresponding security access using their passwords and system login time. Consequently, instead of written access control policies, trained SVMs are used for access decisions.

Cappelletti et al. [30] experiment with multiple ML techniques, including Decision Tree [113], Random Forest (RF) [23], Support Vector Machines (SVM) [36], and Multi-Layer Perceptron (MLP) [120], for making access control decisions. The authors train the ML models using access history. The authors suggest using neural networks for complex systems. The experimentation is performed on two Amazon Datasets [10,11] and the Incident dataset [9]. The details of the datasets are reported in Table 2.1.4, 2.1.8, 2.1.13.

Khilar et al. [83] propose a trust-based approach that allows access to the cloud resources based on the accesses history and access behavior of the user. Also, the approach considers other factors, including user behavior, bogus request, unauthorized request, forbidden request, and range specifications.

Srivastava et al. [127] propose a novel access control framework named risk adaptive access control (RAdAC), which understands the genuineness of the requester, calculates the risk, and then acts accordingly. The framework considers many real-world attributes in its design, such as time of access, location of access, frequency of the requests, and the sensitivity of requested resources. The proposed approach depends on multiple parameters such as a sensitivity score and relevance, and the authors used unique strategies to engineer those parameters. The authors developed a prototype of their proposed method for a Hospital Management System (HMS). They experiment with a neural network with two hidden layers and an RF algorithm.

The authors in [93] propose an Efficient Permission Decision Engine scheme based on Machine Learning (EPDE-ML) that improves the policy decision point (PDP) of the ABAC model. Internally, the EPDE-ML includes an RF algorithm trained based on user attributes and prior access control information. For any access control request from a user, the decision engine results either permit or deny indicating the user has access or not to the corresponding resource.

There are some challenges with supervised learning approaches in the real-world system, such as limited labeled data, sparse logs holding partial access information, etc. To tackle these issues, Karimi et al. [77] propose an adaptive access control approach that learns from the feedback provided by the user. The authors propose a reinforcement learning system in which an authorization engine adapts an ABAC model, as shown in Figure 2.4. The model depends on interacting with

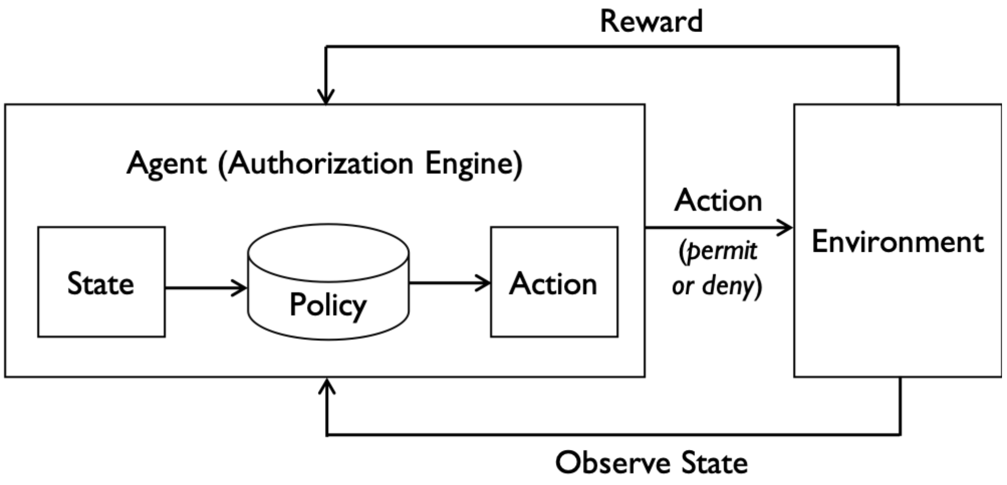

**Figure 2.4**: Overview of Reinforcement Learning Based ABAC Policy Learning.

the system administrator to receive the feedback that helps the model make an authorization decision. The authors suggested four methods for initializing the learning model based on attribute value hierarchy to speed up the learning process. The authors experiment on multiple data sets, including synthesized and real ones, to evaluate the proposed method properly. The synthesized authorization records are produced based on ABAC policies: a set of ABAC policies with manually written policy rules and one with randomly generated policy rules. The real dataset is built from the records provided by Amazon [10] (Table 2.1.8).

**Table 2.5**: Summarizing Machine Learning for Access Control Decision

| Reference | Application | Problem Considered | Access Control Model | ML Approach | Dataset Type |
|---|---|---|---|---|---|
| Chang et al. 2006 [33] | Not specified | A novel access control model with time constraint | Time-constraint Access Control | SVM | Syn |
| Khilar et al. 2019 [83] | Cloud Computing | Policy for cloud resources based on the access history and behaviour | Trust-Based Access Control | RF, DT, SVM, Neural Network, etc. | Not Specified |
| Cappelletti et al. 2019 [30] | Not specified | Inferring ABAC policies from access logs | ABAC | DT, RF, SVM, MLP | RW (Table 2.1.4, 2.1.8, 2.1.13) |
| Srivastava et al. 2020 [127] | Defense, airport, and healthcare | A novel access control framework to decide accesses based on the genuineness of the requester | Risk Adaptive Access Control (RAdAC) | Neural Network, RF | Not Specified |
| Liu et al. 2021 [93] | Big Data & IoT | Improves the policy decision point (PDP) of the ABAC model | ABAC | RF | RW (Table 2.1.8) |
| Karimi et al. 2021 [77] | IoT | Adaptive ABAC policy learning | ABAC | Reinforcement Learning | Syn & RW (Table 2.1.8) |

# CHAPTER 3: OPERATIONAL MODEL OF MACHINE LEARNING BASED ACCESS CONTROL

This chapter proposes DLBAC, a new approach to automated and dynamic next-generation access control, and develops a candidate DLBAC model, DLBAC$_\alpha$. Experimental results demonstrate that DLBAC$_\alpha$outperforms classical policy mining and machine learning techniques in many aspects, including capturing the existing access control state of the system accurately and generalizing well to situations that were not seen during training time. Also, as DLBAC is a neural network, we address previously highlighted concerns on the explainability of the black-box nature of neural network-based systems for access control [30]. We apply deep learning interpretation methods to confirm that it is possible to understand DLBAC decisions in human terms to a large degree (albeit not with 100% accuracy). Next, we synthesize several large-scale access control datasets with a varying number of users and resources. We evaluate the performance of DLBAC on synthetic and two real-world datasets. The outcome of this research has been published at the following conference:

- Mohammad Nur Nobi, Ram Krishnan, Yufei Huang, Mehrnoosh Shakarami, and Ravi Sandhu. "Toward Deep Learning Based Access Control." In Proceedings of the Twelveth ACM Conference on Data and Application Security and Privacy (CODASPY), Baltimore-Washington DC Area, United States, 2022.

## 3.1 Introduction and Motivation

There are various mainstream access control models available, including Access Control Lists (ACLs) [58], Role-Based Access Control (RBAC) [118], and Attribute Based Access Control (ABAC) [64]. These approaches have dominated the domain with excellent performance and tremendous progress has been made over time [79]. However, one basic issue has remained the same for over forty years. Skilled security administrators needed to engineer and manage accesses as only humans could develop detailed policy insights about individuals' needs within the broader organization. Clearly, this leads to all types of errors and inefficiencies [16]: there remain plenty of users with accesses that should not have those accesses (over-provisioned to ease administrative burden) and plenty of users that lack accesses that should indeed have those accesses (under-provisioned for the sake of tightened security) [49, 125]. Administrators tactfully perform a balancing act to maximize security and minimize costs. This complexity is further exacerbated with the proliferation of cloud-based applications that perform machine-to-machine access through APIs, Internet of Things (IoT), Bring Your Own Devices (BYOD), etc. Also, the engineered rules in traditional access control policies typically make poor access control decisions for user-resource metadata that were not explicitly seen by the mining process.

To solve these problems, we propose an automated and dynamic access control mechanism leveraging advances in deep learning technology [120] that could complement or potentially replace existing traditional access control systems. It can significantly reduce the effort and involvement of a human administrator in maintaining an access control system. This approach denoted as Deep Learning Based Access Control (DLBAC), addresses four major limitations, including attribute engineering, policy engineering, policy and attribute update, and generalization, of classical access control approaches such as RBAC and ABAC. Without loss of generality, we use the term attribute to refer to any form of traditional access control information such as roles and relationships.

## 3.2 Classical Access Control Approaches vs. DLBAC

In this section, we provide a brief overview of DLBAC and explain how it differs from classical approaches.

### 3.2.1 Decision Making in Classical Approaches vs. DLBAC

Figure 3.1 illustrates how DLBAC makes a decision as compared to two classical access control approaches (the discussion applies to other forms of access control approaches such as relationship based access control or ReBAC [34]). In RBAC, an access control decision is simply a cross-

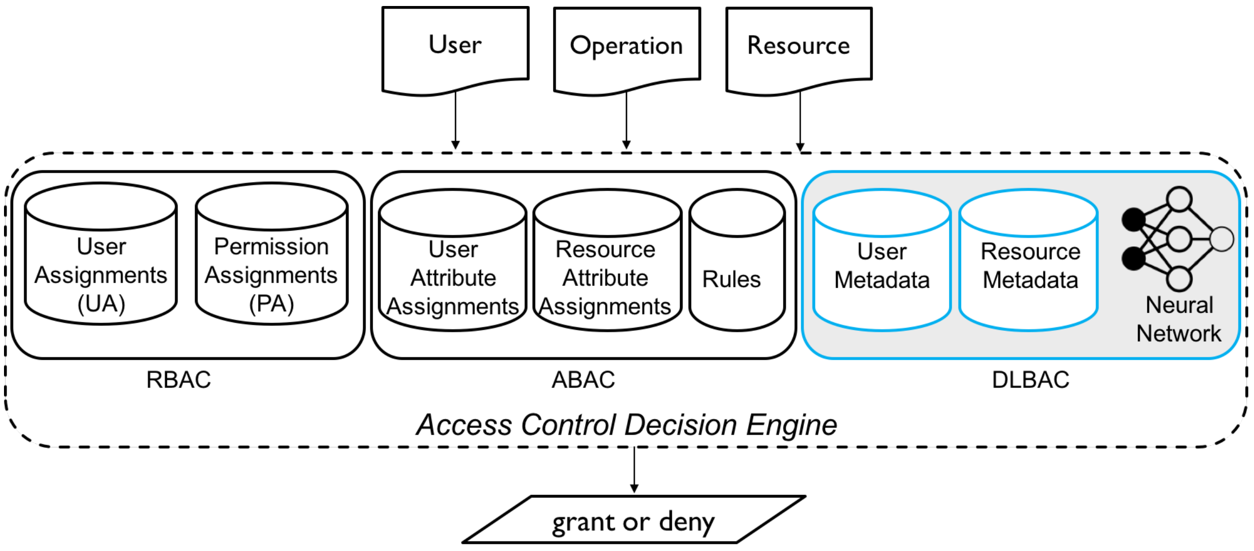

**Figure 3.1**: Decision Making in Classical Approaches vs. DLBAC.

reference between user-role and permission-role assignment relations. In the case of ABAC, an access control rule is evaluated for a given operation based on the attributes of the user and resource in question (sometimes attributes of other entities such as "environment" are used as well). In DLBAC, a deep neural network makes an access control decision based on the available *metadata* for the user and resource. For example, metadata could include logs of accesses, employee join date, access time, network access profile, etc. For simplicity, we assume metadata are represented as name-value pairs. While syntactically they appear to be the same as attributes, which are often name-value pairs as well, semantically they are very different. Metadata are primarily different from attributes since they do not go through the access control design and engineering process. A typical organization could host multiple applications such as email, file storage, human resources, benefits, and other cloud services. Each of those applications hold metadata about users and resources in the organization. Metadata are designed inherently as part of the functionality engineering phase of the system instead of during the access control design phase of the system. Therefore, they are immediately available to DLBAC once the system is implemented. For example, 'join_date', 'spending_history' and 'credit_history' could be metadata of customer, whereas an engineered attribute could be 'status' (such as 'status = platinum'), determined based on all of those metadata.

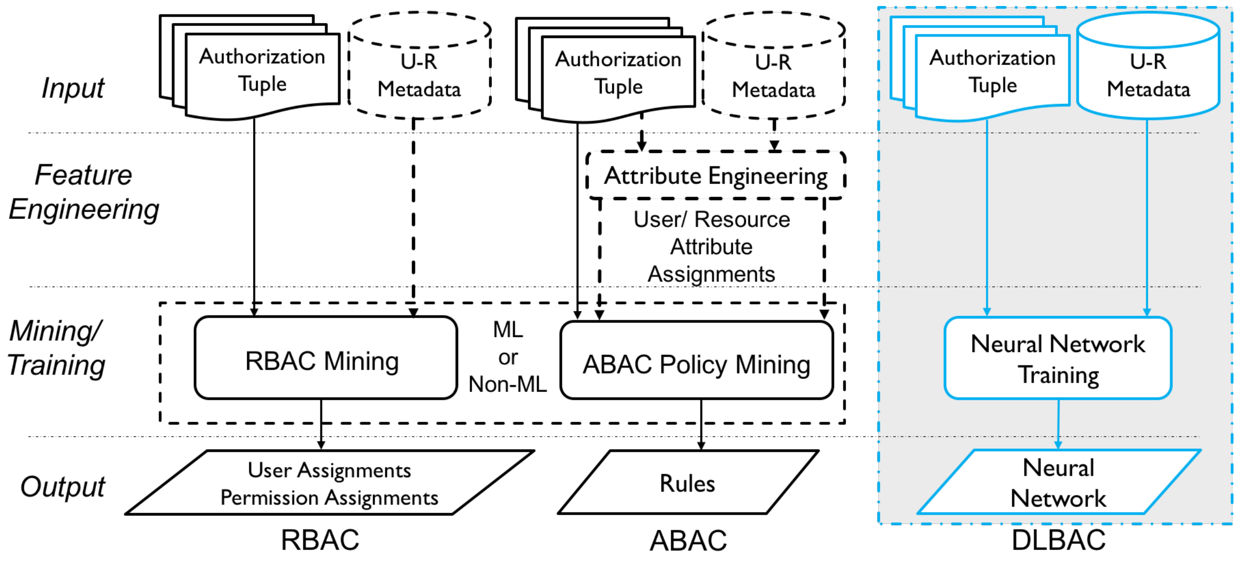

**Figure 3.2**: Design Process of Classical Approaches vs. DLBAC.

### 3.2.2 Policy Engineering in Classical Approaches vs. DLBAC

A conceptual representation of classic access control approaches versus DLBAC is depicted in Figure 3.2. For simplicity, we assume all approaches obtain the current access control state of the system as authorization tuples (e.g., ⟨user, resource, operations⟩), and the metadata of users (e.g., ⟨designation, "employee"⟩) and resources (e.g., ⟨size, "small"⟩) as the input. In ABAC, the first step of *attribute engineering* involves designing attributes of users and resources in the system that are selected and properly assigned based on available metadata. Common to ABAC and RBAC, the second stage is *policy mining/engineering*, through which proper policies are developed. Access control mining algorithms, including those using machine learning (ML), are summarized in Section 2.3. The last element in the conceptual representation of models is the output. For the RBAC approach, the mining process's output is a set of roles, permission assignment to roles (PA), and user assignment to roles (UA). For ABAC, the output includes a policy consisting of a set of access control rules. (Note that most ABAC mining works assume that attributes and attribute assignments are already available [105, 115]). In contrast, DLBAC is an end-to-end access control approach. It does not need feature engineering since it works directly with the metadata of users and resources. The DLBAC approach's output is a trained deep neural network, which takes user and resource metadata as input and makes access control decisions. We note that the DLBAC is agnostic to any deep neural network architecture.

## 3.3 DLBAC$_\alpha$: A Candidate DLBAC Model

This section presents a prototype of DLBAC, namely DLBAC$_\alpha$, which is an access control model built upon the proposed DLBAC approach in Section 3.2. As illustrated in Figure 3.2, DLBAC models such as DLBAC$_\alpha$ need to be fed with authorization tuples and user/resource metadata. We apply DLBAC$_\alpha$ to two real-world and eight synthetic datasets. First, we describe how we construct our synthetic datasets and introduce the real-world datasets. Then, we discuss how the DLBAC$_\alpha$ neural network is trained, and access decisions are made.

### 3.3.1 Synthetic Dataset Generation

An approach to generate synthetic access control datasets was proposed in [143]. We adopt this approach with minor changes to generate multiple synthetic datasets. We briefly discuss this approach here.

The algorithm first generates a set of attribute names for users and resources randomly. Next, it generates a set of synthetic *rules* based on those attributes and then uses these rules to inform user/ resource creation and attribute value assignments. Each *rule* is a tuple of the form $\langle UAE, RAE, OP, C \rangle$, where $UAE$ is the set of user attribute expression, $RAE$ is the set of resource attribute expression, $C$ is the set of constraints, and $OP$ is a set of operations. For example, *rule(title=student; type=document; read; department=department)* is a sample synthetic rule where *title=student* represents the UAE, *type=document* represents the RAE, *read* is the operation, and constraint is denoted by *department=department*. A user will be authorized to operate on a resource if the user satisfies the UAE, the resource satisfies the RAE, and both the user and resource satisfy the constraint stated in that rule. For each rule, the algorithm generates a set of users that satisfy the rule and then generates resources where for each resource, there is at least one user available to satisfy the rule. The following user and resource are created based on the above rule: *user(student1, title=student, department=cs)*, *resource(document1, department=cs, type=document)*. Here, the *student1* and *document1* are the unique ids of a user and a resource, respectively. The user *student1* has two attributes (*title* and *department*), and the resource *document1* also has two attributes (department and type). Also, student1 satisfies UAE, as the user has the title *student* which is part of the title in UAE. Similarly, document1 satisfies RAE for the *type* attribute. Both *student1* and *document1* satisfy the rule's constraint as they are from the same department. Thus, according to this rule, *student1* has *read* access to *document1*.

Finally, once the rules are generated, users and resources are created, and attributes are assigned, it is straight-forward to create the authorization tuples. For each user, resource and operation combination that satisfies a rule, an authorization tuple is created or updated with a new operation, as the case may be.

## Syntax of Synthetic Dataset

We adapt this data generation approach by creating many metadata instead of attributes. We maintain four operations and various metadata (eight to thirteen) for each user/resource for different datasets. We define the syntax of DLBAC$_\alpha$'s dataset to contain a set of authorization tuples. An authorization tuple could be illustrated of the form $\langle uid|rid|m^u_1 : v_1, m^u_2 : v_2, ..., m^u_i : v_i|m^r_1 : v_1, m^r_2 : v_2, ..., m^r_j : v_j|\langle op_1, op_2, op_3, op_4\rangle\rangle$. The uid and rid in the tuple indicates the unique id of a user and a resource, respectively. The next part gives the metadata values of all $i$ metadata of a user and $m^u_1$ indicates the first user metadata name (e.g. umeta0) whereas its value is indicated by $v_1$. The subsequent part presents the metadata values of all $j$ metadata of a resource, and first resource metadata name (e.g. rmeta0) and its value are represented by $m^r_1$ and $v_1$, respectively. The last part is a binary sequence with a '1' meaning 'grant' and a '0' meaning 'deny' for that operation. For example, $\langle$1011|2021|*30 49 5 26 63 129 3 42* | *43 49 5 16 63 108 3 3* |$\langle$*1 1 0 1*$\rangle\rangle$ is a sample authorization tuple of our dataset where 1011 and 2021 are the user and resource's unique number. The next eight numbers indicate the metadata values of a user, the following eight numbers represent resource's metadata values, and the final four binary digits signify the user has

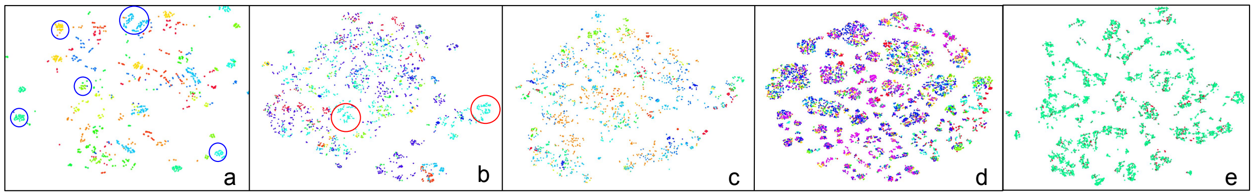

**Figure 3.3**: Comparing Complexity of Datasets. (a) 1650 users, 320 resources, eight user/resource metadata, (b) 1000 users, 639 resources, 11 user/resource metadata, (c) 800 users, 656 resources, 11 user/resource metadata, (d) 4500 users, 4500 resources, 11 user/resource metadata, (e) An Amazon dataset [10] (The dataset has more samples with 'grant' access. Therefore, for better visualization, we considered all the tuples with 'deny' access permission and randomly selected a similar number of tuples with 'grant' access permission).

$op_1, op_2, op_4$ access to the resource. (In the example above, we also skip naming the metadata with the assumption that it could be inferred from the position of the metadata value.) For simplicity, we assume the metadata values in our datasets are categorical, and each metadata value is an *integer* representation of a category. We anticipate that our results will hold even in cases of metadata with *real numbers*.

### Dataset Visualization

We use t-SNE plots [135] to visualize the samples in our datasets. A t-SNE plot discovers relationships in the data by identifying analogous clusters of data points with several features and projecting high dimensional features into a low dimensional feature space, while retaining essential information of the data. We project each of our samples to a 2-dimensional feature and plot them. (Note that our datasets have varying numbers of features/metadata ranging from 16 to 26 in total.) Each *dot* in the plot (Figure 3.3) represents an authorization tuple, where multiple tuples of the same color indicate that they have the same access permissions. For example, two tuples with only *read* and *write* access permissions will have the same color. It is worth mentioning that a dataset with n operations will have tuples with $2^n$ different access combinations and can be plotted with $2^n$ distinct colors. For instance, the authorization tuples of a dataset with two operations (e.g., read and write) can be plotted with four different colors (tuples with the read access, tuples with the write access, tuples with both read and write access, tuples with no access).

The position of a tuple in the plot is fixed according to the user and resource metadata values. For instance, two different users with the same metadata values may have access to a resource or multiple resources having the same metadata values. Therefore, these tuples will have the same position regardless of their access permissions. Figure 3.3(a) depicts a dataset of 1650 users and 320 resources. The tuples (dots) take the position all over the plot according to their metadata values. This dataset has four different operations, and thereby, there are tuples with 16 distinct colors. However, we observe different tuples with the same access permission (same color) are grouped, and groups are isolated from one another, as shown in the figure with blue circles. Thus, a simple classifier can easily distinguish them without much difficulty (e.g., making one rule for each circle). The access control states in real-world situations might be much more complicated [102]. Figure 3.3(e) shows the visualization of a dataset from Amazon. Even though this is not a complete access control scenario of the entire Amazon enterprise [37], samples with access (green dots) significantly overlap with other samples without access (red dots) which means tuples with very similar metadata values have entirely different accesses. A simple classifier would create too many rules to model such a dataset.

### Introducing Complexity into Synthetic Datasets

Informed by t-SNE visualization of the Amazon dataset, we seek to introduce complexity into our synthetic datatsets to closely reflect real-world situations. We observe that, in practice, the access privileges of users and resources with somewhat similar metadata could vary. It is also expected that not all the metadata of a user/resource would contribute equally to their permissions. To reflect such scenarios in some of our datasets, we determine accesses in tuples by considering *all* the metadata values of user and resource but *hide* a portion of metadata values from the policy miner (when dealing with mining approaches) and model training phase (when dealing with machine learning approaches). However, during rule generation, we ensure that the metadata that we will hide contribute to a lesser extent toward permission decisions by excluding them from being part of the constraint of the rules. In a nutshell, hiding metadata attempts to simulate a scenario, where access decisions are made based on access control *attributes* that are not fully informed by the entire metadata set. In a perfect world, access control attributes could capture the relevant metadata distributed across an organization succinctly. However, it is reasonable to hypothesize that this is not a practical assumption.

Figure 3.3(b) represents a dataset with 11 user metadata and 11 resource metadata. The authorization tuples were created considering all the eleven metadata. To simulate a similar situation while visualizing tuples and understand how it looks from a policy mining (or classification) perspective, we make only the first eight metadata values of the user and the first eight metadata values of the resource available for visualization. As depicted in the figure, different dots of other colors now start to mix, indicating that two tuples with very similar user-resource metadata values may have very different accesses. Indeed, such proximity of the tuples is challenging for clustering or rule creation. The more metadata we hide, the more complicated the dataset will be for policy mining algorithms and machine learning approaches.

Note that we still notice a few portions in the plot where the same colored dots are clustered together and hence separable, as shown with red circles in Figure 3.3(b). This is because the dataset generation algorithm [143] creates the metadata values based on a distribution, where the value range (i.e., number of values for each metadata in the dataset) is sparse. For example, there are around one hundred different values for specific metadata (e.g., *department*) in a dataset with hundred users. One can easily cluster all the users into hundreds of groups based only on the department. However, there might be the case where metadata values are required to be chosen from a set of a limited number of values (say, ten departments for hundred users). Evidently, it is harder to cluster one hundred users into ten groups than a hundred. To reflect this, for each metadata, we define a fixed and smaller set of values (6 to 20 unique values) following the same distribution used by Xu et al. [143]. We choose each metadata value from the corresponding list during user/resource creation and metadata value assignment. This strategy creates datasets with significantly overlapped samples as depicted in Figure 3.3(c). We also extended the number of users and resources to simulate a larger organization, which adds more overlaps among samples

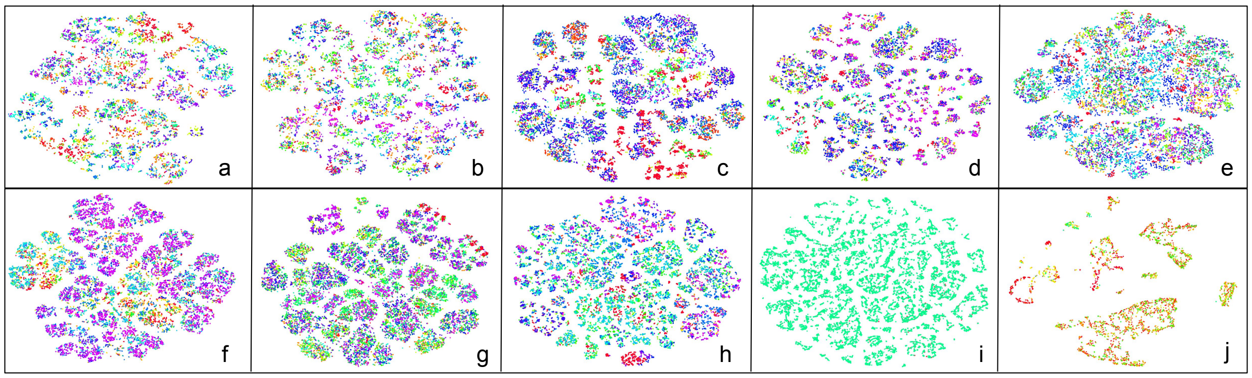

**Figure 3.4**: t-SNE Visualization of Synthetic and Real-world Datasets from Table 3.1. Figures a-h corresponds to synthetic datasets #3−#10 and i-j corresponds to real-world datasets #1−#2 in Table 3.1, respectively.

**Table 3.1**: List of Datasets.

| # | Dataset | Type | Users | User Metadata | Resources | Resource Metadata | Authorization Tuples |
|---|---|---|---|---|---|---|---|
| 1 | *amazon-kaggle* | Real-world | 9560 | 8 | 7517 | 0 | 32769 |
| 2 | *amazon-uci* | Real-world | 4224 | 11 | 7 | 0 | 4224 |
| 3 | $u4k$-$r4k$-$auth11k$ | Synthetic | 4500 | 8 | 4500 | 8 | 10964 |
| 4 | $u5k$-$r5k$-$auth12k$ | Synthetic | 5250 | 8 | 5250 | 8 | 12690 |
| 5 | $u5k$-$r5k$-$auth19k$ | Synthetic | 5250 | 10 | 5250 | 10 | 19535 |
| 6 | $u4k$-$r4k$-$auth21k$ | Synthetic | 4500 | 11 | 4500 | 11 | 20979 |
| 7 | $u4k$-$r7k$-$auth20k$ | Synthetic | 4500 | 11 | 7194 | 11 | 20033 |
| 8 | $u4k$-$r4k$-$auth22k$ | Synthetic | 4500 | 13 | 4500 | 13 | 22583 |
| 9 | $u4k$-$r6k$-$auth28k$ | Synthetic | 4500 | 13 | 6738 | 13 | 28751 |
| 10 | $u6k$-$r6k$-$auth32k$ | Synthetic | 6000 | 10 | 6000 | 10 | 32557 |

and higher complexity to the dataset, as shown in Figure 3.3(d).

Finally, we synthesized eight different datasets (datasets #3-#10 in Table 3.1) used for DLBAC$_\alpha$ experimentation and evaluation, with varying numbers of users, resources, user and resource metadata, and authorization tuples, each reflecting a varying degree of complexity. We use the following naming convention for our synthetic datasets, as listed in Table 3.1: $u\langle$approx. number of users$\rangle$ − $r\langle$approx. number of resources$\rangle$ − $auth\langle$approx. number of authorization tuples$\rangle$. We use 'u', 'r', and 'auth' to indicate users, resources, and authorization tuples, respectively. We also visualize all the synthetic datasets in Figure 3.4(a-h) using t-SNE plots. As illustrated in the Figure, each plot has dots with one of 16 different colors (for four operations), and those dots mix extensively.

### 3.3.2 Real-world Dataset

Amazon published two datasets that contain access control information which is widely used in access control research [30, 37, 76]. We name these datasets as $amazon\text{-}kaggle$ and $amazon\text{-}uci$, and list them in the Table 3.1 (1-2). The $amazon\text{-}kaggle$ dataset was released in Kaggle [10] (a platform for predictive modeling competitions) as a challenge to the community to build a machine learning model to determine the employees' accesses. The dataset holds historical access data where employees were manually allowed or denied access to resources over time. The dataset has about nine thousand users and seven thousand resources with over 32K authorizations tuples. Each tuple specifies eight user metadata that depicts a user's properties, a resource id to identify the resource, and a binary flag to indicate whether the user has access to the resource or not. However, the dataset is highly imbalanced, and about $93\%$ of the tuples are with grant access. We visualize this dataset in Figure 3.4(i). As we see, there are dots from two colors where green and red correspond to tuples with grant and deny access, each. Notably, a significant number of dots are from grant accesses.

The $amazon\text{-}uci$ dataset was provided by Amazon in the UCI machine learning repository [11]. This dataset contains access information of more than 36,000 users and 27,000 permissions. For any permission, less than $10\%$ of all users have requested access. The dataset is widely used in access control researches, and in most of the cases, the experiments are confined to only 5 to 8 most requested permissions [30, 37, 76]. Likewise, we took the seven most requested permissions, and for each permission, we list users who have access to the selected permissions. However, this dataset is also imbalanced, around $75\%$ tuples with the deny access. In addition, the dataset is not ABAC in nature, and there are some tuples in the dataset where users with identical attribute values do not have the same access permissions. Because of this, policy mining or classification approaches may suffer while clustering the users for different permissions. We visualize this dataset in Figure 3.4(j).

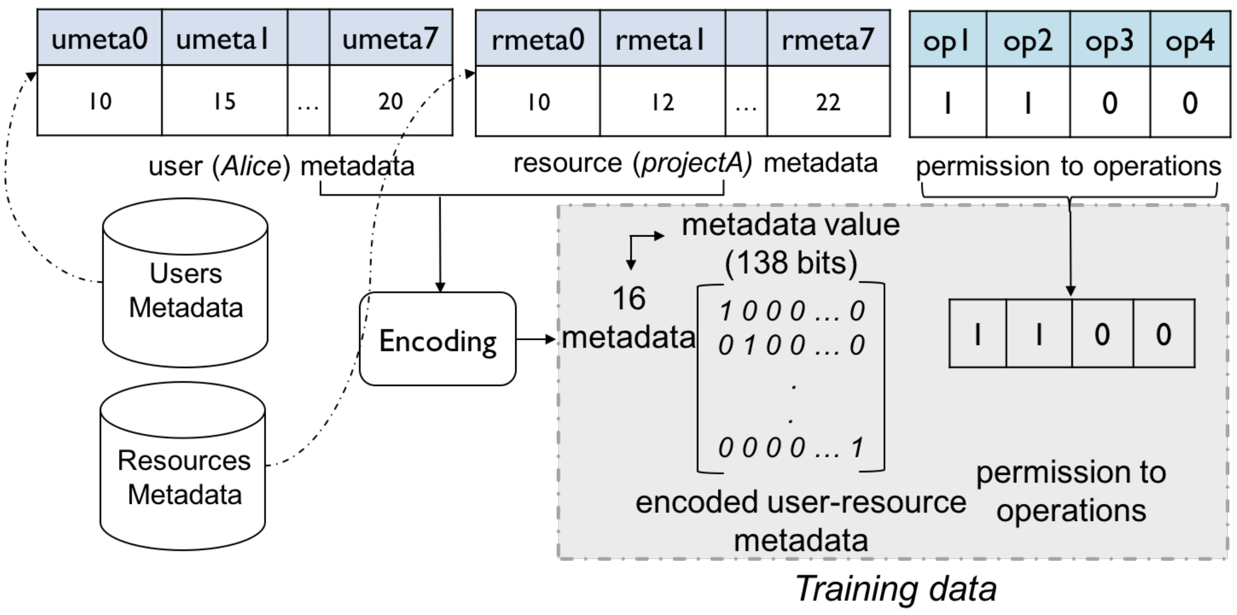

**Figure 3.5**: Preparing Training Data for DLBAC$_\alpha$.

## 3.3.3 Neural Network Architecture and Training

For DLBAC$_\alpha$, the deep neural network takes user/resource metadata values as input. It includes a classification layer with the number of neurons equal to the number of operations, where each neuron outputs the *probability* of granting the permission for a related operation, *op*. Given a feature vector $x$ of the user and resource metadata, the neural network can be defined as a prediction function $f$ such that $\hat{y} = f(x)$, where $\hat{y}$ is the predicted label or permission (grant (1) or deny (0)) of *op*, obtained from comparing the probability of granting the permission at the output of the network with a *threshold*. Note that for all DLBAC$_\alpha$ experimentation, we consider a threshold of 0.5. To train the neural network $f$ (i.e. determine the network's weights), a set of training authorization tuples $\mathbb{X}$ of size $N$ is collected, where $(x_i, y_i)$ denotes the $i$-th sample in $\mathbb{X}$, where $x_i$ is the feature vector of the user and resource metadata and $y_i$ is the corresponding $op$ or the target label. As discussed in Section 3.3.1, for some synthetic datasets, we hide a portion of metadata from the policy mining algorithm to mimic some complex situations. We apply this for the datasets having more than eight user metadata and eight resource metadata. In such a case, $x_i$ represents the feature vector of the first *eight user metadata* and the first *eight resource metadata*. So, for example, for

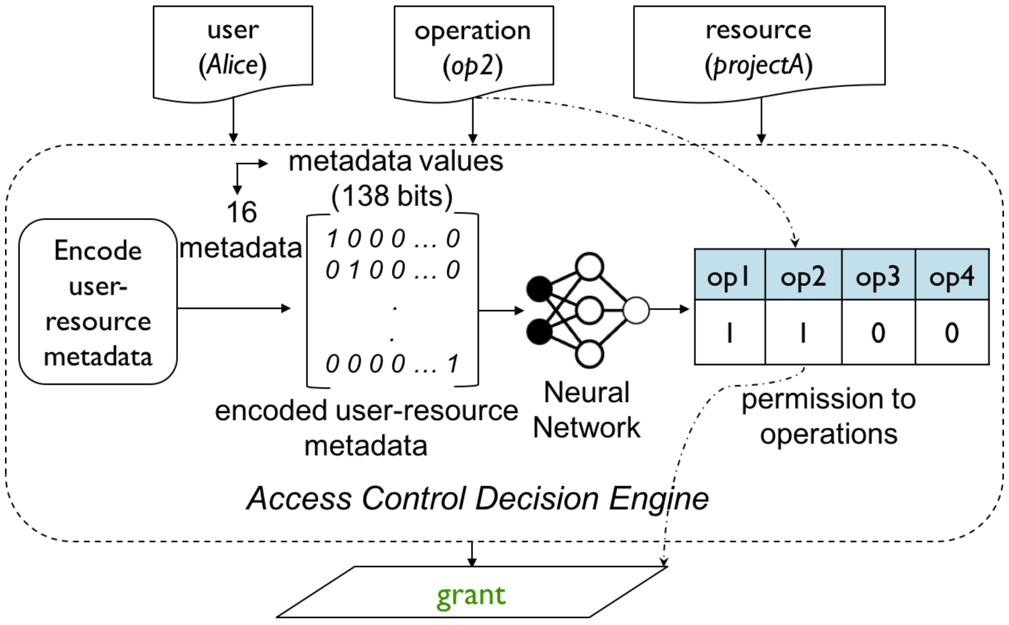

**Figure 3.6**: Decision Making Process in DLBAC$_\alpha$.

a dataset with 13 user-resource metadata, we hide five metadata from the user metadata and five from the resource metadata.

Since the metadata values in our dataset are categorical, we map them to numerical or binary values by utilizing embedding or encoding. We encode a tuple's user/resource metadata value using *one-hot encoding* [57] to transform the categorical values into a two-dimensional binary array. The row in the array represents metadata, and the column holds the encoded binary representation of the corresponding metadata value. (Amazon dataset's metadata values are too sparse, and we use *binary encoding* for them considering its memory efficiency in such cases [121]). As each operation is already binary, we apply them directly as the target labels without any processing. Figure 3.5 illustrates the overall training data preparation for DLBAC$_\alpha$. As illustrated, we encode the metadata values of a user $Alice$ and a resource $projectA$ and apply permissions to the associated operations without further processing. We train DLBAC$_\alpha$ based on the training data, and this *trained* DLBAC$_\alpha$ is used in *Access Control Decision Engine* (discussed below) to produce access decisions for test data.

### 3.3.4 Decision Making Process in DLBAC$_\alpha$

*Access Control Decision Engine (Decision Engine)* is a DLBAC component responsible for receiving and authorizing any access request. In DLBAC$_\alpha$, the *Decision Engine* (Figure 3.6) takes three inputs (*user, resource, and operation*). The *Decision Engine* retrieves the user and resource metadata from the internal databases and then encodes them to obtain corresponding binary representation. The encoded input is fed into the neural network to predict the corresponding request's access permission. The network outputs access information for all the operations. The decision engine then determines the actual access authorization based on the requested access and the network's output. For the specific example in Figure 3.6, user *Alice* wants *op2* access on *projectA* resource. The output of the neural network for the *op2* is $1$, which indicates that *Alice* has *op2* access on *projectA*. Thus, the *Decision Engine* authorizes this request.

## 3.4 Evaluation

In this section, we experimentally evaluate the performance of DLBAC$_\alpha$ using both synthetic and real-world datasets.

### 3.4.1 Evaluation Methodology

We experiment and evaluate the performance for all the datasets listed in Table 3.1. We consider each dataset to represent an organization with its own unique characteristics. We split each dataset into training (80%) and testing (20%) sets. As the test dataset is entirely *unseen* during training, the evaluation shall adequately measure the generalization of any method.

**Instances of DLBAC$_\alpha$.** DLBAC is agnostic to any deep neural network architecture, and we will show that the performance of deep learning-based models is consistent across datasets. As part of the demonstration, we implement three instances of DLBAC$_\alpha$ using three distinct deep neural network architectures including ResNet [59], DenseNet [69] and Xception [35], and we name the instances as DLBAC$_{\alpha-\mathrm{R}}$, DLBAC$_{\alpha-\mathrm{D}}$ and DLBAC$_{\alpha-\mathrm{X}}$, respectively. For DLBAC$_{\alpha-\mathrm{R}}$, we use the ResNet architecture with depth 8 for the first four datasets in Table 3.1 whereas, for the rest of the datasets, we use a ResNet with depth 50. For DLBAC$_{\alpha-\mathrm{D}}$, we use the DenseNet architecture with [6,12,24,16] layers in the four dense blocks. For all the model architecture implementation, we adopt the source code from Keras application.[1]

The DLBAC$_{\alpha}$ instances were developed in Python using Keras library with a TensorFlow backend and trained on Google Colab (a 12GB NVIDIA Tesla K80 GPU). We apply Adam optimizer with an initial learning rate of 0.001, scheduled to reduce the learning rate by dividing by ten after every 10 epochs. The epoch and batch size was chosen as 60 and 16, respectively, with an early stop after 5 consecutive epochs without any performance improvement. As the DLBAC$_{\alpha}$ outputs an access probability between '0' and '1' for each operation, we use binary cross-entropy loss. We have created a repository in GitHub consisting of all the datasets, source code, and trained networks.[2]

**Machine learning (ML) algorithms.** We compare the performance of DLBAC$_{\alpha}$ instances with classical machine learning approaches such as Support Vector Machine (SVM) [36] and Random Forests (RF) [23]. We also compare with Multi-Layer Perceptron (MLP) [120] with four hidden layers (non-deep) to evaluate how significant the performance difference is between a deep and a non-deep neural network. We use the *SVC* and *RandomForestClassifier* class of the Python scikit-learn library [110] for SVM and RF implementation, respectively, with their default configurations. We implement MLP using Keras library.

**Policy mining algorithms.** There is no other existing *deep learning-based* access control approach to the best of our knowledge, so a direct comparison of our work results is not currently possible. Therefore, we compare DLBAC$_{\alpha}$ with ABAC policy mining algorithms being one of the flexible and generalized access control approaches. We compare the performance of DLBAC$_{\alpha}$ instances with the following policy mining algorithms. While a few other works exist as discussed in our related work, a key decision factor in selecting these works was our ability to readily access their source codes and our ability to clearly understand, modify/tweak as needed and compile them.

1. The policy mining algorithm proposed by Xu and Stoller [143], which we refer to as XuS-

[1]`https://github.com/keras-team/keras-applications`
[2]`https://github.com/dlbac/DlbacAlpha`

toller.

2. Rhapsody [37], a policy mining algorithm built upon an ML algorithm for subgroup discovery named APRIORI-SD [81]. Rhapsody performance has a direct correlation with multiple parameters. We experimented with different parameter values and selected the policy with the highest F1 score while maintaining an FPR below 0.05.

3. EPDE-ML [93], a permission decision engine scheme based on machine learning where a trained RF model makes the access control decision.

**Evaluation metrics.** For ML algorithms, we compute the F1 score and compare the performance with DLBAC$_\alpha$ instances and show that deep learning based algorithms generally perform better than traditional ML and MLP techniques. For an extensive comparison against policy mining algorithms, we compute the F1 score, False Positive Rate (FPR), True Positive Rate (TPR), and Precision. We consider the standard definitions [37, 93] of these evaluation metrics. Policies (or models) with a higher F1 score lead to better generalization. They can make more accurate access control decisions on users and resources with attributes not explicitly seen during the mining (or training) process. Also, the higher TPR and Precision are better as these scores indicate how accurately and efficiently the policies (or models) can grant access. On the contrary, the policies (or models) with a lower FPR are better as they are less likely to give access to requests, those which should be denied according to the ground truth access control policy.

### 3.4.2 Results

#### Performance comparison with ML algorithms

Figure 3.7 illustrates the overall performance of all ML approaches and DLBAC$_\alpha$ instances for each dataset with respect to F1 score. The performance of all the algorithms is consistent and better for the $amazon$-$kaggle$ dataset, but it is not the case for $amazon$-$uci$ dataset. In this case, SVM and MLP performed significantly lower, and other approaches including DLBAC$_\alpha$ instances could not achieve high performance. Such a performance is expected due to the inconsistency in the access

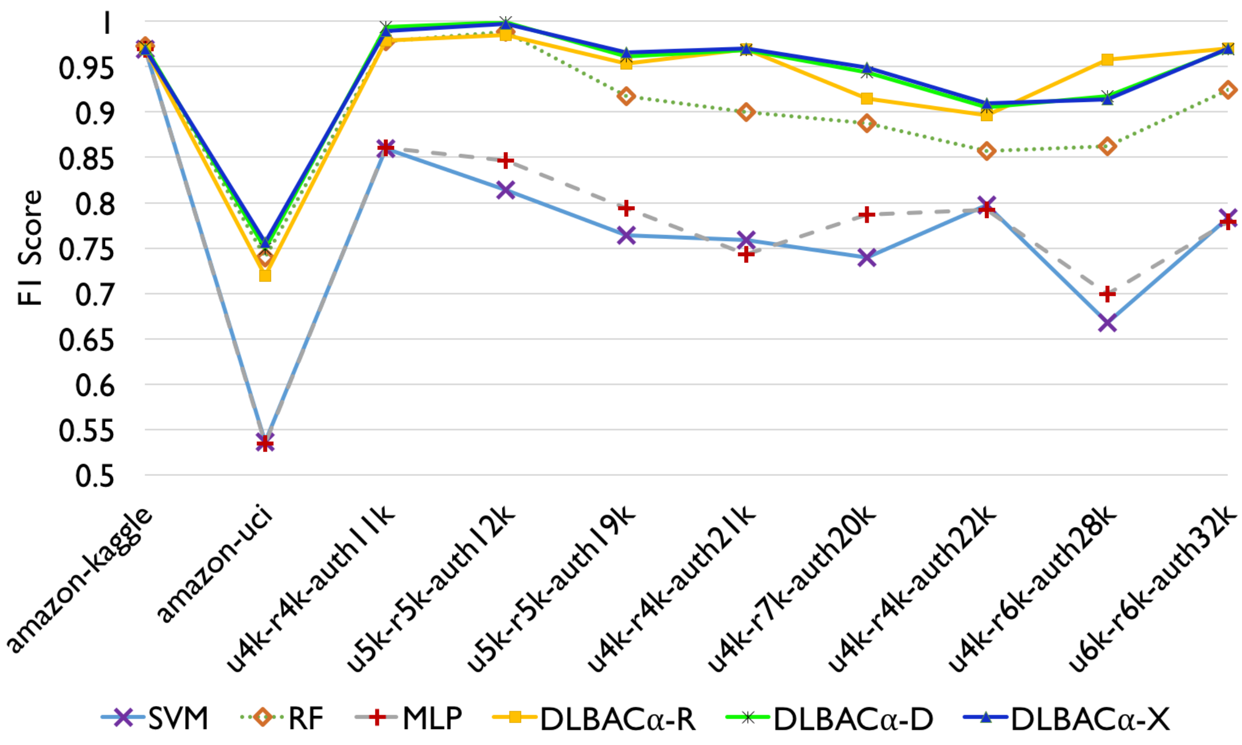

**Figure 3.7**: F1 Score Comparison: ML Algorithms vs. DLBAC$_\alpha$ Instances.

permissions in that dataset. The dataset contains tuples where users with identical attribute values do not have the same access permissions, as discussed in Section 3.3.2. The advantage of the DLBAC approach is more evident when the dataset if properly processed. We show this using our synthetic datasets. For synthetic datasets, DLBAC$_\alpha$ instances achieved the highest F1 score for all datasets, while SVM and MLP perform the worst. Also, the instances of DLBAC$_\alpha$'s improvements over RF are significant (for p-value $< 0.05$; paired T-test (not shown here)) for all the synthetic datasets except $u5k$-$r5k$-$auth12k$ dataset. The performance advantage of DLBAC$_\alpha$ instances is particularly pronounced in synthetic datasets with a large number of authorization tuples, where DLBAC$_\alpha$ instances report 0.03 to 0.09 improvements over RF as shown in the figure. Notably, the performances of all algorithms vary with the complexity of the datasets. However, DLBAC$_\alpha$ instances show the lowest variation in its performance across the datasets, suggesting that DLBAC$_\alpha$ is most robust against changes in data characteristics such as number of hidden metadata, number of users and resources and authorization tuples counts. Except for the $u4k$-$r6k$-$auth28k$ dataset, all algorithms' performances drop by increasing the number of hidden metadata (e.g., the $u4k$-

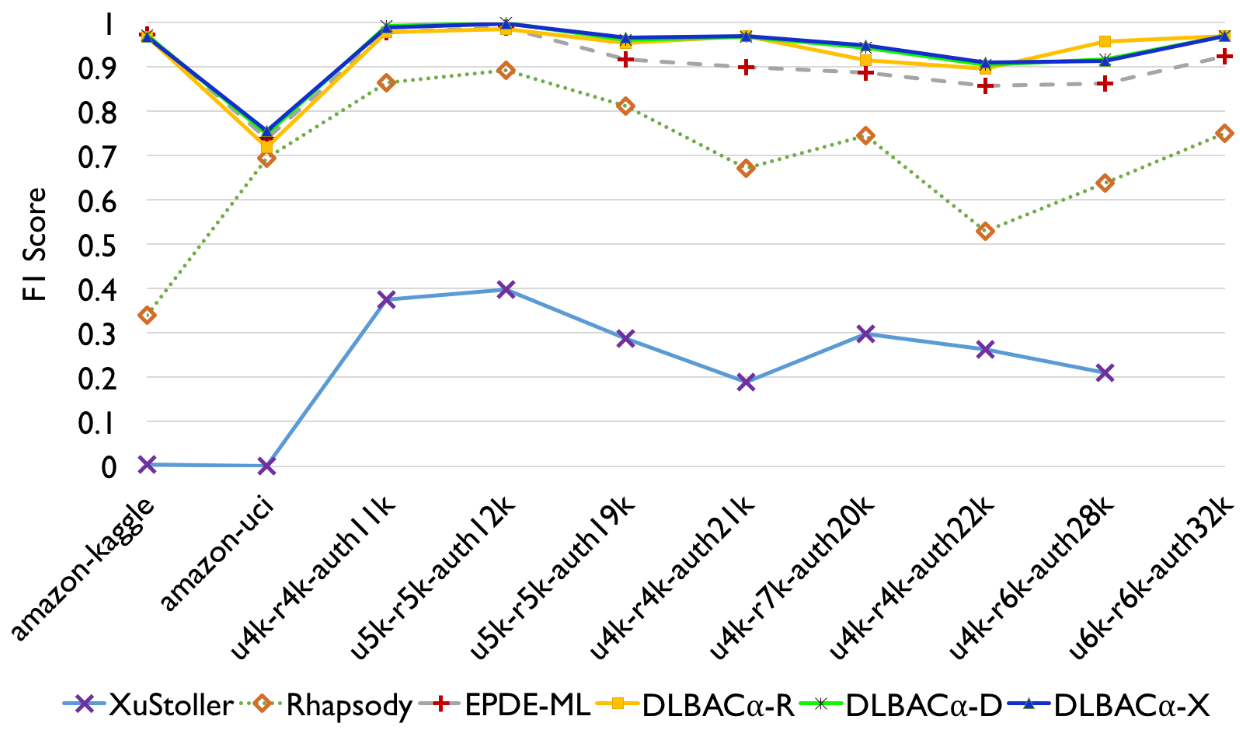

**Figure 3.8**: F1 Score Comparison: Policy Mining Algorithms vs. DLBAC$_\alpha$ Instances.

$r4k$-$auth22k$ dataset with 13 metadata where we hide 5 of the metadata from the feature vector as discussed in Section 3.3.3), suggesting that an increase in data complexity generally impacts performance. Overall, the experimental results indicate that DLBAC$_\alpha$ is more effective and robust than classical machine learning approaches, including MLP, for making accurate access decisions.

While the performance advantages of DLBAC$_\alpha$ are not apparent in the Amazon datasets, we emphasize that those datasets are not entirely reflective of the access state complexity of the entire organization but that of a small portion of the company. This is one of the reasons why we synthesized additional datasets for our experimentation.

**Performance comparison with policy mining algorithms**

Figures 3.8, 3.9, 3.10, and 3.11 compare the F1 score, FPR, TPR, and Precision, respectively of policy mining algorithms and DLBAC$_\alpha$ instances. We could not experiment XuStoller algorithm for the largest synthetic dataset ($u6k$-$r6k$-$auth32k$) as it took a very long runtime without any output. We can make the following observations based on all these experimental results.

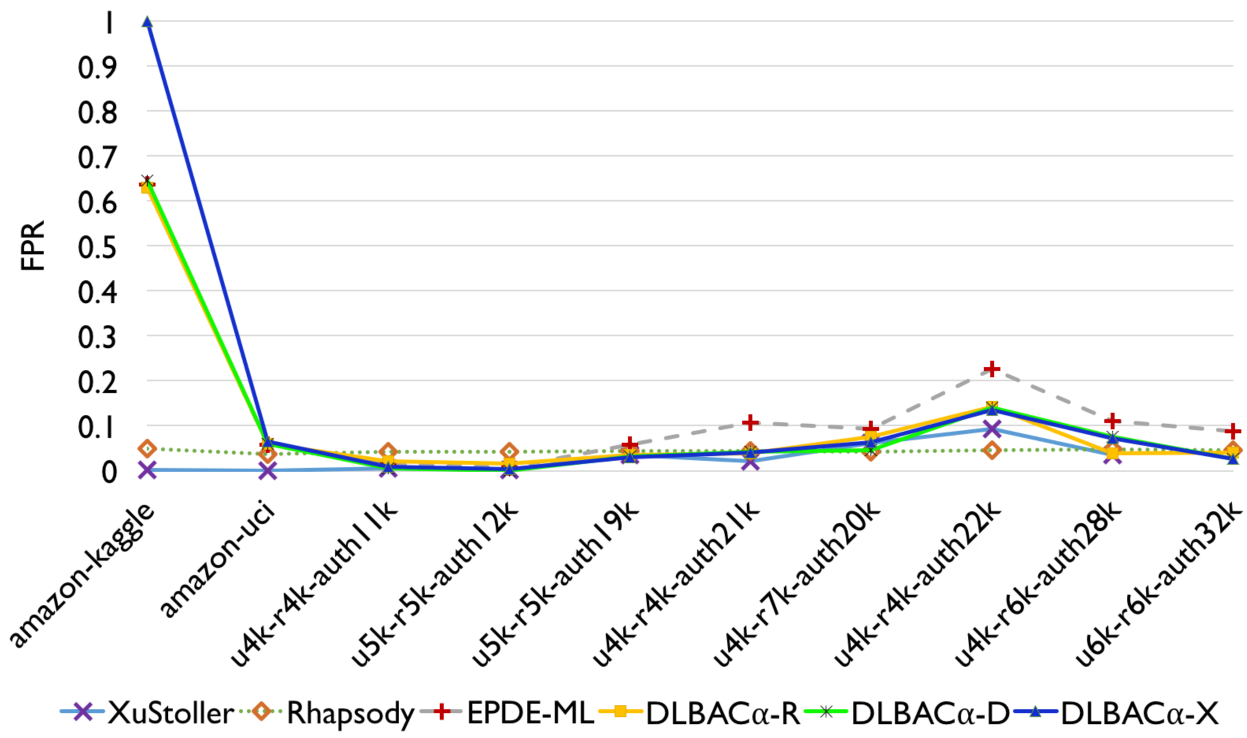

**Figure 3.9**: FPR Comparison: Policy Mining Algorithms vs. DLBAC$_\alpha$ Instances.

- *A deep learning based approach can make more accurate access control decisions and generalize better.* The F1 score of EPDE-ML and DLBAC$_\alpha$ instances are significantly better than the rule-based approaches such as XuStoller and Rhapsody. That signifies, in general, machine learning-based approaches can make better generalization and accurate access control decisions. As we see in Figure 3.8, the performance improvements of DLBAC$_\alpha$ instances over EPDE-ML, which is statistically significant for most datasets, suggest that deep learning based algorithms make the more accurate decision and have even better generalization capability than classical ML-based policy mining approaches.

- *A deep learning based approach can properly balance both over-provision and under-provision.* XuStoller and Rhapsody achieved the best FPR as shown in Figure 3.9, indicating they are unlikely to give access to requests that should be denied according to the actual access control policy. However, that is not the case for denying access. As we see in Figure 3.10, the TPR for Rhapsody is between 0.2 to 0.85, and for XuStoller, it is from 0 to 0.25. Such a lower TPR indicates that these algorithms are pretty inefficient while denying access (under-

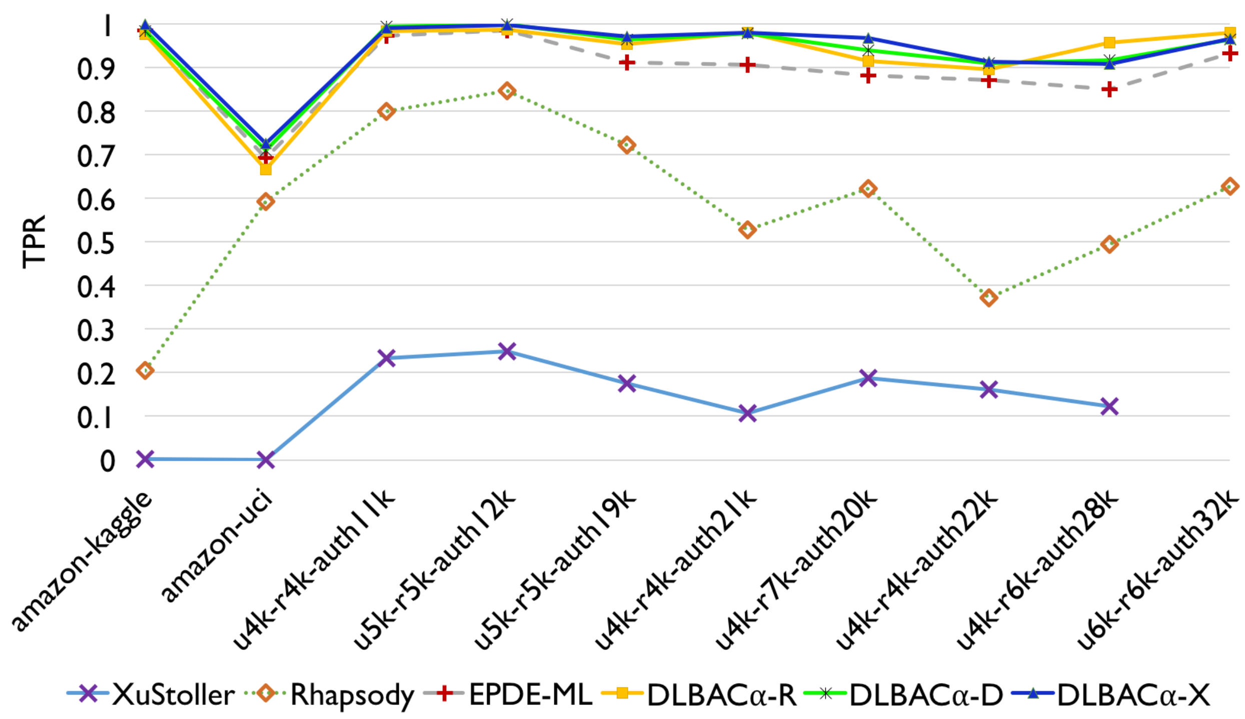

**Figure 3.10**: TPR Comparison: Policy Mining Algorithms vs. DLBAC$_\alpha$ Instances.

provisioned) even though the requests deserve grant access according to the actual access control policy. On the other hand, the EPDE-ML performed lowest in terms of FPR (over-provisioned) across synthetic datasets ranging from 0.06 to 0.23. Their average TPR and Precision are below 0.9. Such higher FPR and comparably lower TPR and Precision suggest that EPDE-ML could not achieve desirable performance in terms of over-provision and under-provision. The DLBAC$_\alpha$ instances obtained a much higher TPR and Precision as illustrated in Figure 3.10 and 3.11. Also, the DLBAC$_\alpha$ instances reached an FPR which is comparable to XuStoller and Rhapsody, as shown in Figure 3.9. This suggests that deep learning based approach can balance better between over- and under-provisioning.

- *An imbalanced dataset may affect some performance that could be calibrated.* The FPR result of DLBAC$_\alpha$ instances for $amazon\text{-}kaggle$ dataset is high, and the TPR for $amazon\text{-}uci$ dataset is relatively low, arising due to the characteristics of the dataset [137]. As discussed in Section 3.3.2, these datasets are imbalanced, one has unreasonably more samples from the grant class, and the other has more from the deny class. We argue that this is a typi-

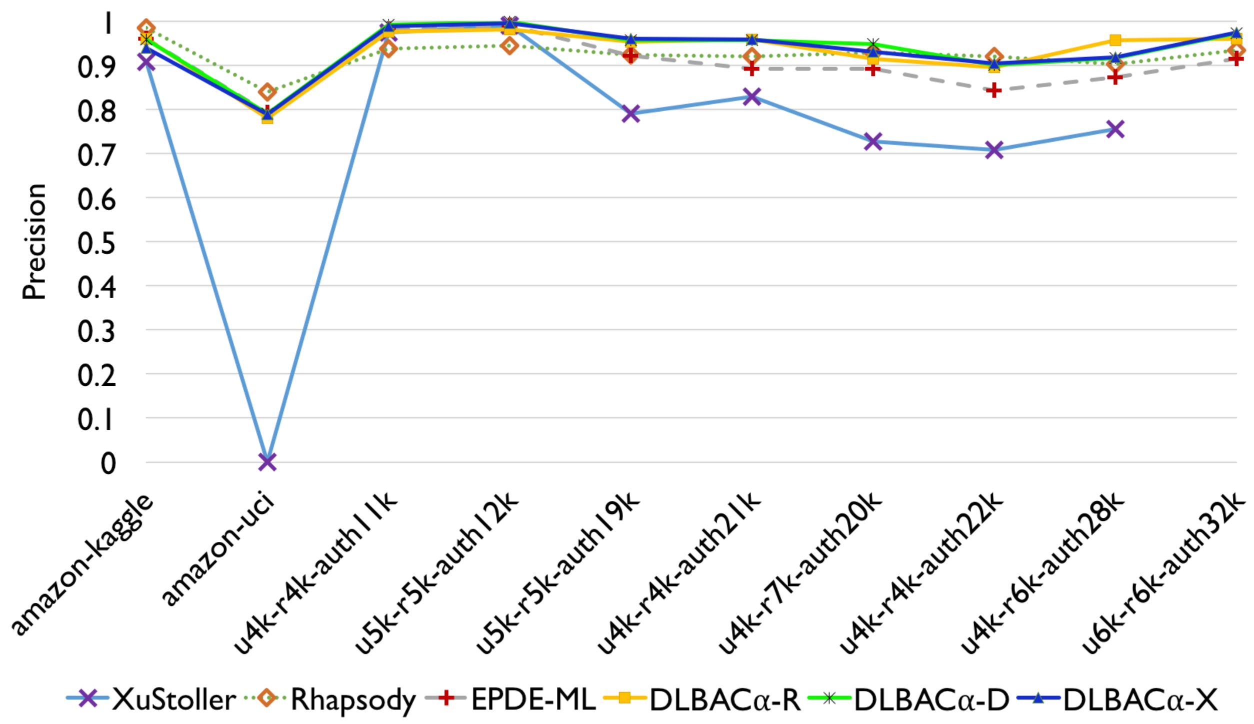

**Figure 3.11**: Precision Comparison: Policy Mining Algorithms vs. DLBAC$_\alpha$ Instances.

cal machine learning problem, and EPDE-ML has a similar performance for these datasets. For balanced datasets, the FPR and TPR of DLBAC$_\alpha$ instances are consistent, and FPR is below 0.05 for most of the datasets while TPR is above 0.95, as demonstrated in Figure 3.9 and 3.10. Evidently, these metrics could be calibrated based on the tolerance to over vs. under-provisioning of an application context. Specifically, one could favor a particular metric in DLBAC$_\alpha$ by modifying the loss function to increase the weight of the minority class [63] (discussed next), adjusting the threshold for granting permissions as described in Section 3.3.3, etc. We note that an improvement in one of the metrics will likely negatively impact one or more of the others and that every work is prone to this issue.

**Improving FPR Performance.** We apply a well-known technique for performance improvement (or control) of an imbalanced dataset. The idea is to give more weight to the wrong prediction for samples from minority classes and vice-versa. However, improving in one metrics might affect the other performances. We experiment with DLBAC$_{\alpha-\mathrm{D}}$ instance for the *amazon-kaggle* dataset where there are two different classes for each sample, deny and grant. In the *amazon-kaggle*

**Table 3.2**: Different Combination of Weight for Different Classes for the Loss Function.

| Combination Name | Grant Class Weight | Deny Class Weight |
|---|---|---|
| equal weight | 1.0 | 1.0 |
| wlc-1 | 0.07 | 0.93 |
| wlc-2 | 0.03 | 1.0 |
| wlc-3 | 0.01 | 0.8 |
| wlc-4 | 0.01 | 1.0 |

dataset, there are only below 7% of the samples from denying class. Hence, we modify the loss function while training the DLBAC$_{\alpha-D}$ to add majority weights on loss for the deny class and significantly smaller weights for the other one. As reported in Table 3.2, we consider four different combinations of weight for the loss function for different classes. Figure 3.12 showed, with the weight decrease for the grant class and increased weight for the deny class, the FPR reduces. It also positively impact precision which rises consistently. However, the F1 score and TPR are also affected and decrease gradually.

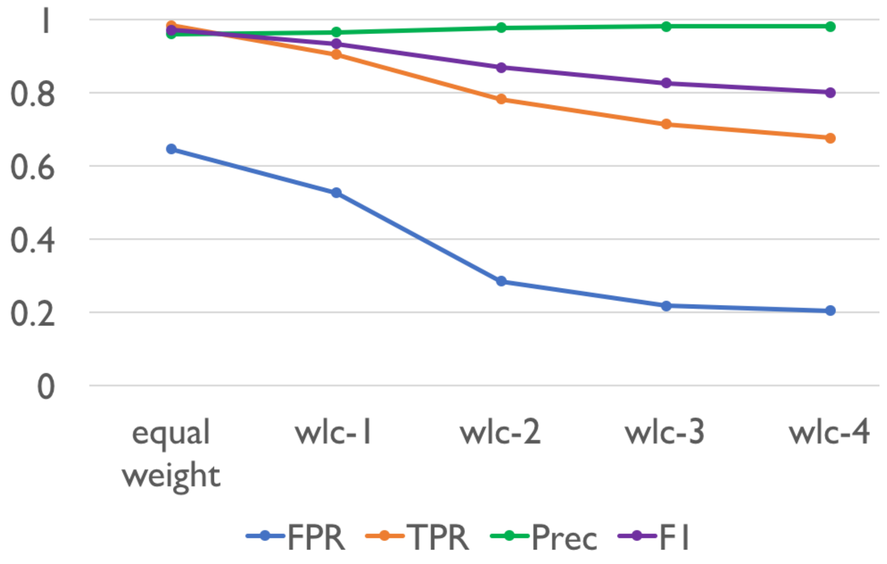

**Figure 3.12**: FPR Performance Improvement in DLBAC$_{\alpha}$.

Overall, the DLBAC$_{\alpha}$ instances achieved better or comparable performance for all the metrics across datasets, suggesting that deep learning based approaches generalized better and made more accurate access control decisions than rule based and classical ML based approaches. These results demonstrate the effectiveness of using DLBAC as an access control system.

## 3.5 Understanding DLBAC Decisions

As the core of a DLBAC system is a *neural network*, a major challenge is providing insights into why and how DLBAC makes certain decisions. That is, explainability is a key challenge for DLBAC. For instance, in Figure 3.6, the *Decision Engine* received a request that user *Alice* wishes *op2* access on *projectA* resource. Based on the result of the *neural network*, the decision engine approved the request. However, it is not quite obvious why the *neural network* made that prediction for this request. Such a justification is generally straightforward in traditional access control systems as the decisions are made based on written policies. But, it is challenging for DLBAC due to the *black-box* nature of a neural network [124]. As the decisions are made based on provided *user/resource metadata*, it is essential to understand why a decision is made and which metadata influenced that decision. Many techniques have been introduced to help gain insights into a neural network's internal details. In this section, we investigate two state-of-the-art approaches for this purpose: *Integrated Gradients* [129] and *Knowledge Transferring* [51]. We experiment on $\text{DLBAC}_{\alpha-R}$ instance (as introduced in Section 3.4.1), and for brevity, we refer it to $\text{DLBAC}_{\alpha}$ in the following discussion. The related source code is uploaded to GitHub.[3]

### 3.5.1 Integrated Gradients

*Integrated Gradients* is an effective interpretation technique that focuses on attributing the decisions of a neural network to the input features of the prediction samples. It attributes a network's decision to its input features in terms of gradient, which specifies the most effective elements for a decision. To understand the *decision* of $\text{DLBAC}_{\alpha}$ for a *tuple*, it is required to provide the *user/resource metadata values*, the *decision*, and the *neural network* as input to the *Integrated Gradients*. Then, *Integrated Gradients* outputs the *attribution scores* of the input metadata that we normalize in the scale of 0 to 1, denoting the degree of impact on the decision. Such an interpre-

[3]`https://github.com/dlbac/DlbacAlpha/tree/main/understanding_dlbac_alpha`

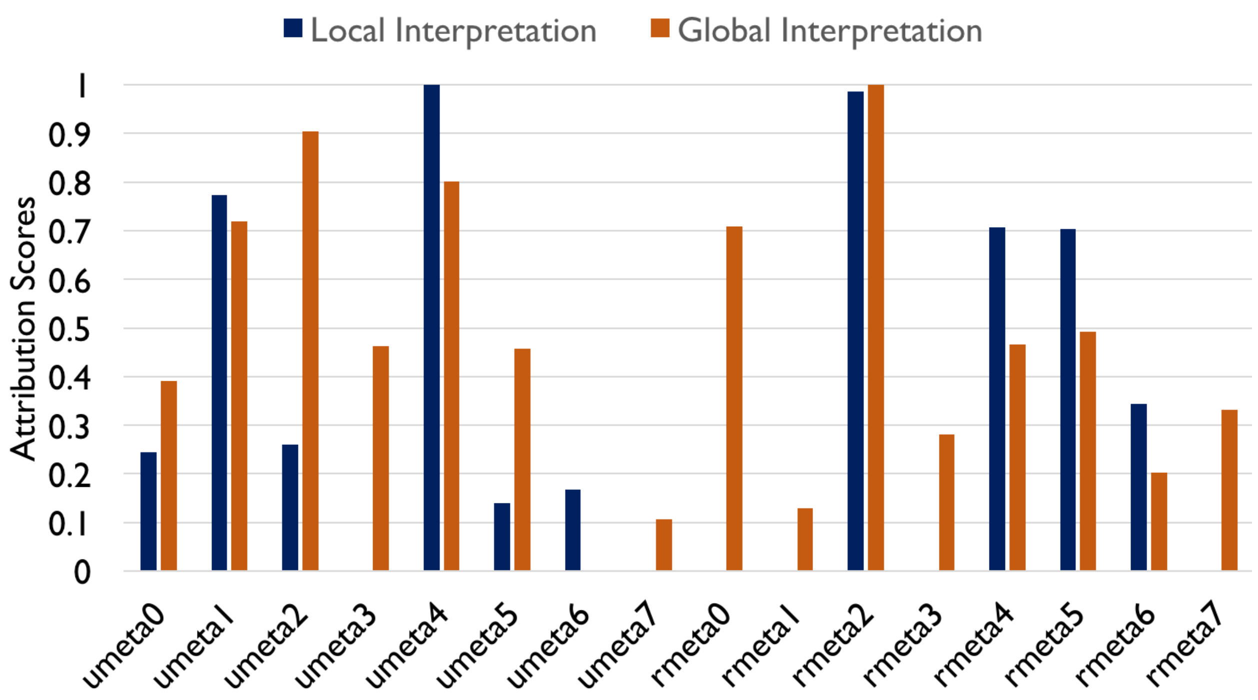

**Figure 3.13**: Local and Global Interpretation (blue: local interpretation of a decision in DLBAC$_\alpha$. orange: global interpretation of DLBAC$_\alpha$ for the grant access of an operation).

tation is known as *local interpretation* which helps to better understand each decision of a neural network. It is also helpful to have a *global interpretation* [54] of a network to understand the network's overall knowledge. Generally, it takes a set of decisions to generate a global interpretation of a network.

For DLBAC$_\alpha$, we investigate explainability for both specific access control decision and the overall knowledge learned by the network. For this purpose, we train DLBAC$_\alpha$ on our $u4k$-$r4k$-$auth11k$ dataset. Then, we request *op1* operation access to *projectD* resource for a user *Dave*. In this case, the Decision Engine grants the request. To learn the reason behind this decision, we perform local interpretation with metadata values of *Dave* and *projectD*, the decision (*grant* access on op1 operation), and the *DLBAC$_\alpha$ network*. As depicted in Figure 3.13 (blue bars), for this particular request, user's *umeta4* and resource's *rmeta2* metadata play the most critical role in granting the *op1* operation access.

To achieve global interpretation for the *grant* access to *op1* operation, we take the *op1* access information for a set of fifty random samples with grant access from the $u4k$-$r4k$-$auth11k$ dataset.

(Note that the more samples we use, the more precise the result we get. However, Integrated Gradients is not memory efficient, so we could not test more samples using our workstation with 16 GB of memory). We provide each user/resource metadata and their corresponding decisions for the *op1* operation to *Integrated Gradients* to determine the global interpretation. Figure 3.13 (orange bars) depicts the global interpretation of DLBAC$_\alpha$ for the $u4k$-$r4k$-$auth11k$ dataset for the *grant* access to *op1* operation. The result identifies the most influential metadata as the resource's *rmeta2*, and the *attribution scores* of other metadata. The second most important metadata is *umeta2* metadata of user.

### Application of Integrated Gradients based Understanding

Improved explainability could be utilized to achieve other benefits by involving humans in the decision-making process. For instance, developers can utilize interpretation techniques to debug [107] incorrect decisions from the *neural network* in *decision engine*. We demonstrate that *Integrated Gradients* based interpretation can be used to grant/deny access permissions by modifying proper metadata.

**Impact of Local Interpretation.** As illustrated in Figure 3.15 (tuple2), the user *Carol* doesn't have op1 access on *projectC* resource. Applying local interpretation on another *permission tuple* with op1 access (e.g., Dave has the op1 access to projectD resource as shown in Figure 3.15 (tuple1)) revealed the *attribution scores* of different metadata for op1 operation. As circled in tuple1, *Dave*'s 'umeta1' and 'umeta4' are the most dominant metadata for this specific access. To grant Carol's op1 access on projectC resource, we utilize *attribution scores* of tuple1. Our result shows that replacing Carol's 'umeta1' and 'umeta4' metadata value with Dave's metadata value enables Carol's op1 access to projectC resource. As reported, modifying one or two significant metadata changes corresponding access.

The opposite holds for least-significant metadata where modifying multiple metadata could not alter related access. To understand the effect of the metadata having non-significant or low attribution scores from Integrated Gradients, we take the attribution scores for the tuple that we

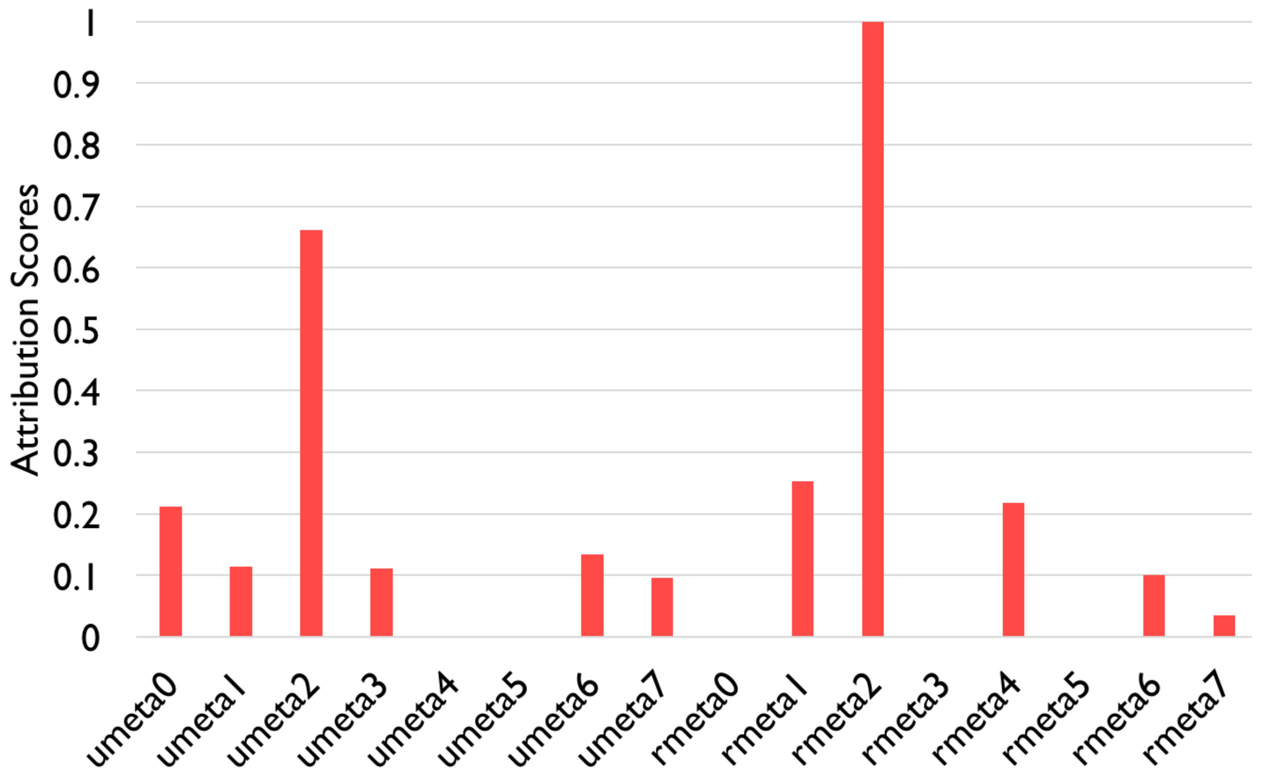

**Figure 3.14**: Local Interpretation of a Tuple with Deny Access.

presented in Figure 3.15 (tuple2). There we see that the user Carol has *deny* access to the projectC resource for op1 operation. We take the attribution scores for the Carol and projectC tuple and, as illustrated in Figure 3.14, the rmeta2 and umeta2 are the most influential metadata with the highest attribution scores. However, umeta4, umeta5, rmeta0, rmeta3, rmeta5 metadata have 0.0 attribution scores, which indicates that they have no impact on this specific decision, whereas rmeta7 and umeta7 (along with some other metadata) have less influence (smaller attribution scores). We want to evaluate the consequence if we change the value of these non or less significant metadata. We modify the value of all these metadata and replace it with the value of a known tuple (Dave and projectD) with grant access. We found no change in the access decision.

**Impact of Global Interpretation.** We also experiment with such a metadata value modification across all the samples of a specific decision (e.g., tuples with 'deny' access on op1) in the same dataset to see the impact of a global interpretation. The idea is to alter the metadata values of influential metadata for a specific decision. For instance, according to Figure 3.13, the *rmeta2* has

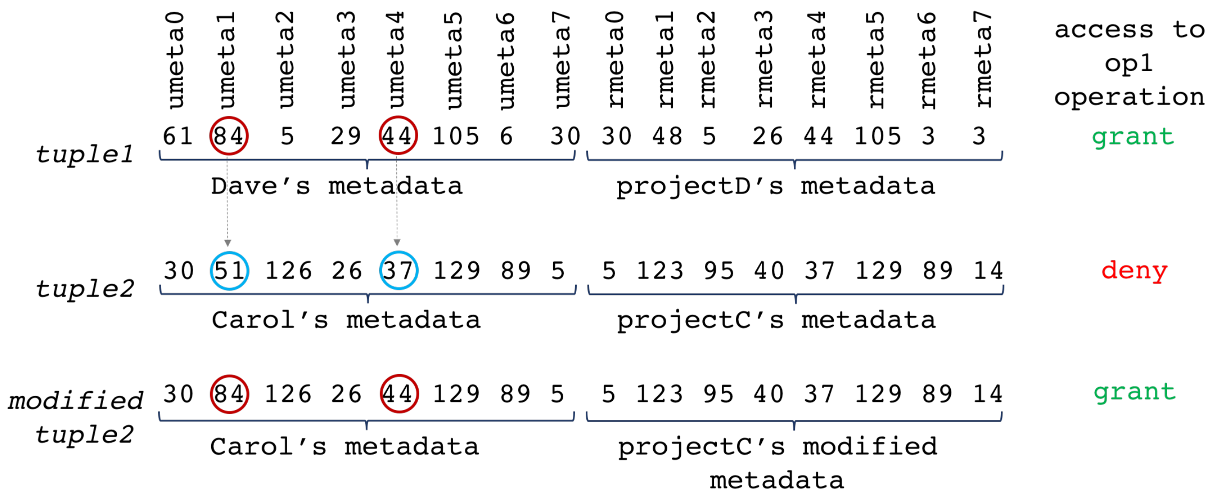

**Figure 3.15**: Applying Integrated Gradients to Grant Access Permission.

the most influence on denying access. If we alter the *rmeta2* value of any tuple, say *tupleA*, with a rmeta2 value from a known tuple with *grant* access, say *tupleB*, the chance of denying access for *tupleA* might reduce and increases the chance of getting access. Also, altering the value of multiple influential metadata of the same tuple may eventually help to get access.

To investigate that we alter different metadata values, one by one, of all the tuples with *deny* access (4581 such tuples) on *op1* in order of their significance level in the global interpretation. For example, we first change the value of rmeta2 metadata, next umeta2 metadata, and so on. We utilize the user/resource metadata values from tuple1 in Figure 3.15 as a known tuple with the 'grant' access for *op1* operation. As described in Figure 3.16, initially, with no change, no tuple has been granted access. However, with the change of first metadata (rmeta2), around 5% of the tuples receive grant access. By changing the second metadata value, around one-third of the tuples get grant access. A similar surge continues to obtain grant access for over half of all the tuples before decreasing for sixth metadata (rmeta5) modification, reaching below 40%. It indicates that some tuples with such a big number of metadata modifications fall under a different distribution for which they have the 'deny' access. Overall, using this technique, a system admin can estimate the impact of metadata value vs. dependent accesses.

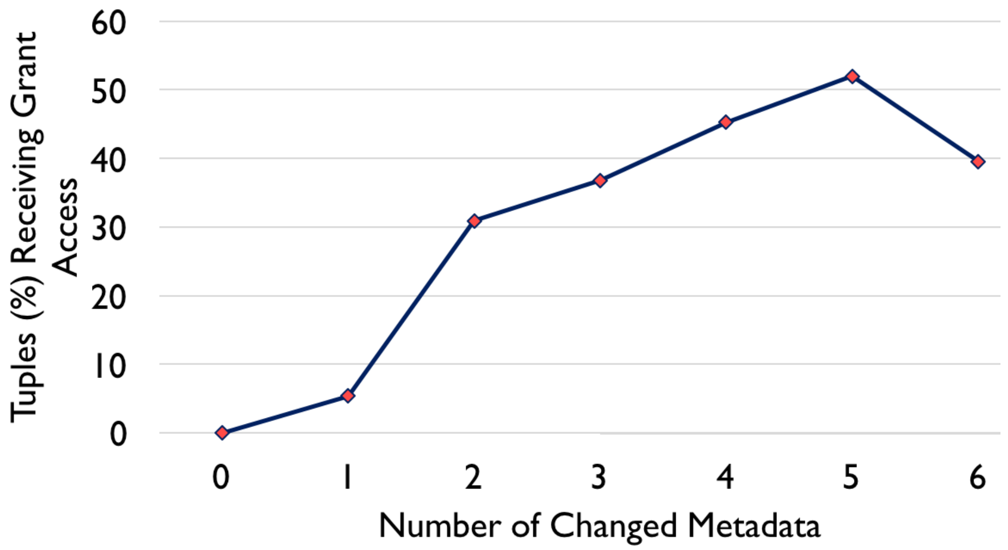

**Figure 3.16**: Modifying Metadata Value Based on Integrated Gradients Across Tuples (4581 tuples) with Deny Access.

### 3.5.2 Knowledge Transferring

Although *Integrated Gradients* determines the *attribution scores* of each metadata for a decision, it does not establish the relationship among metadata or the network's logic [129]. *Knowledge Transferring* could help to identify such relationships. With *Knowledge Transferring*, we can extract a decision tree to approximately understand the decision of DLBAC$_\alpha$ in the form of traditional rules. While accurate reconstruction of the representation details is infeasible, the generated decision tree will give an approximate explanation of the underlying logic of DLBAC$_\alpha$. Though the decision tree makes classification decisions understandable [110], it does not generalize as well as deep neural networks. Interestingly, a neural network's generalization ability is likely to be transferred to a decision tree with the help of a method called *distillation* [62], which has been widely used in ML literature. In this work, we adopt the idea of distillation. As explained in Section 3.3.3, DLBAC$_\alpha$ outputs whether a user has authorization to any resource by giving the *probability* of granting the access instead of a direct *yes/no* answer. Therefore, we can determine those *probability outputs* for all the *tuples* in a dataset and train a decision tree, as shown in Figure 3.17. These probabilities represent the *knowledge* of the neural network, and we aim to transfer it to a decision tree.

To explore *Knowledge Transferring* technique, we train DLBAC$_\alpha$ for the $u4k$-$r4k$-$auth11k$

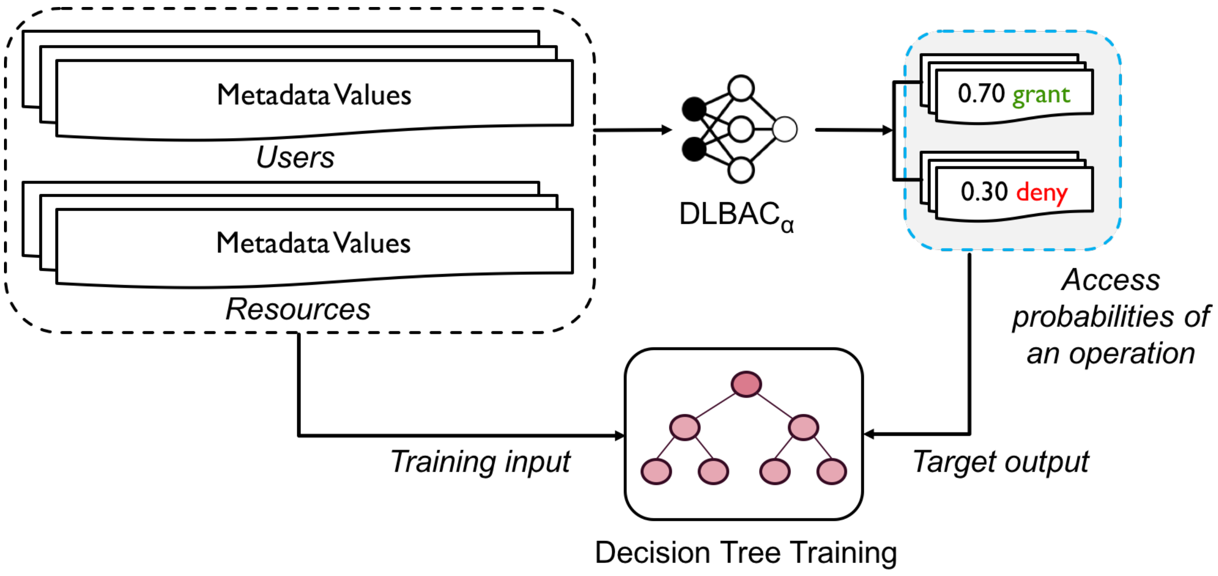

**Figure 3.17**: Knowledge Transferring Technique.

dataset that we used for the Integrated Gradients experiment. We take *op1* access *probabilities* for all the samples. Then, we train a decision tree (DT) on the same dataset. However, instead of giving corresponding ground-truth permissions from the dataset, we provide the *probability* outputs of DLBAC$_\alpha$. Eventually, we construct a DT with a maximum depth of eight that facilitates retrieving underlying rules for any specific decision. We train the tree based on the training data that contain both user-resource metadata and their corresponding access permission for *op1* operation. However, instead of giving permissions from the dataset, we provide corresponding *probabilities* acquired from DLBAC$_\alpha$. We accumulate *op1* access *probabilities* for the tuples in ($u4k$-$r4k$-$auth11k$) dataset, as discussed. Then we train the decision tree based on the training input from the dataset and probabilities as the target output (ground truth).

Figure 3.18 is the decision tree generated from the $u4k$-$r4k$-$auth11k$ dataset (with 8772 samples) based on *Knowledge Transferring* for *op1* operation. We demonstrate how to retrieve a rule from the tree to understand any specific decision. We cropped the tree for better visualization. Each node in the tree represents a binary decision point where the samples are split based on the condition in the corresponding node. For example, the tree's root node is rmeta2 that splits all the 8772 samples based on the 'rmeta2<=17.5' condition. As specified, the metadata values of our

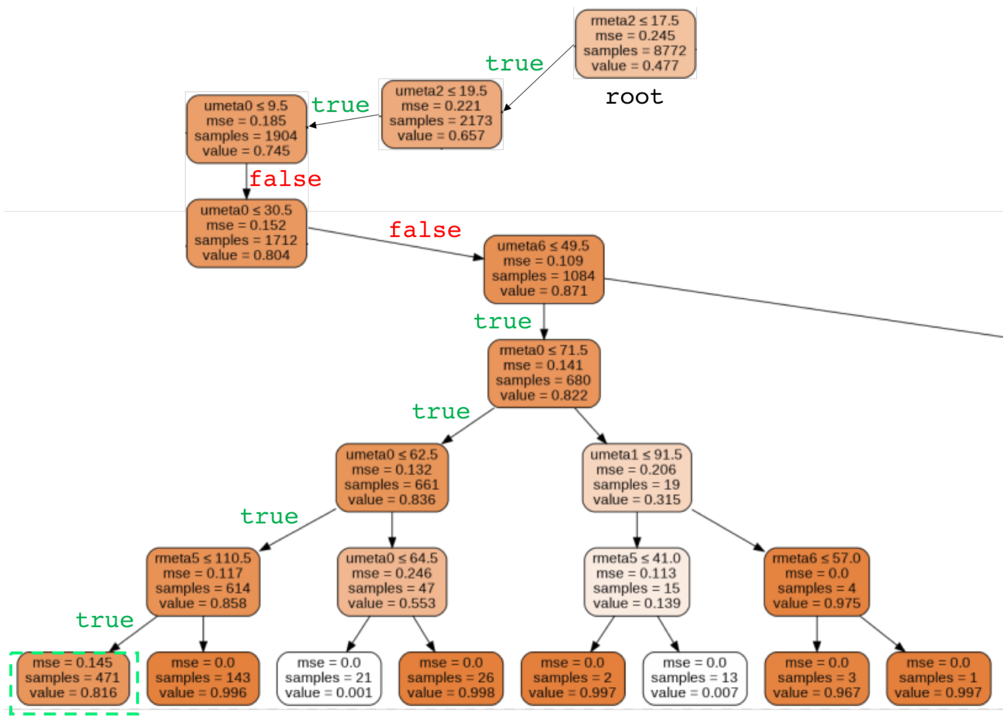

**Figure 3.18**: Decision Tree Generated for $u4k$-$r4k$-$auth11k$ Dataset. Part of the tree has been cropped for better illustration.

datasets are categorical, where values are integer representations of different categories. Therefore, we can round the value in each node's condition to the next integer without any issue. For instance, we can round the value of rmeta2<=17.5 to rmeta2<18 as there is no metadata value of 17.5. The MSE (mean-squared error) indicates the error that measures the quality of a split. Also, the value of the node determines the probabilities for related decisions. As we build this tree for *op1* operation, we have a binary decision point — whether or not the user has access to the corresponding resource for this operation. We round the probabilities to the *grant* if the value is >0.5, otherwise *deny*. As shown in the figure, the value in the root node is 0.478 indicates that initially, all the requests are considered 'denied'.

We assume *Dave* is a user and *projectD* is a resource of the system we considered. We will walk through an access request for user Dave's access to the projectD resource for *op1* operation as described in Section 3.5.2. The metadata value of Dave and projectD is available in Figure 3.15 (tuple1). However, the value of rmeta2 metadata is 5 for projectD; hence the root condition is true, and the next node is the left node with condition umeta2<20. This condition is also successful for

Dave's umeta2=5 and checks the next left node to see the value of umeta0. As the value of umeta0 is 61, it can directly come to the node where umeta6 is verified. This condition is passed with umeta6=6, and the next left condition is also satisfied for the rmeta0=30. The following condition is true as the value of umeta0 is 61. Finally, the tree checks for the value of rmeta5 before making the access decision for Dave to the projectD. This condition is also successful for rmeta5=105. We see the green shaded node is the final decision node for this request. The decision is *grant* based on the value 0.82 (>0.5).

While the tree serves the global interpretation, a rule for any specific decision represents the local interpretation. We retrieve the access rules from the DT for Dave's *op1* access request to projectD resource. We observe that Dave obtained grant access to projectD for op1 based on following rule: $\Big\langle \Big\langle \langle \text{umeta0} > 31 \wedge \text{umeta0} < 63 \rangle \wedge \langle \text{umeta2} < 20 \rangle \wedge \langle \text{umeta6} < 50 \rangle \Big\rangle \wedge \Big\langle \langle \text{rmeta0} < 72 \rangle \wedge \langle \text{rmeta2} < 18 \rangle \wedge \langle \text{rmeta5} < 111 \rangle \Big\rangle \Big\rangle$. It is worth mentioning that the decision tree should be generated with unlimited depth to obtain more precise rules. Note that the Integrated Gradients and Knowledge Transferring techniques are orthogonal in terms of insights they each provide into the neural network and do not substitute each other. For better insights, one could use both methods.

## 3.6 Conclusion

In this chapter, we propose DLBAC, a deep learning based access control approach, to deal with issues in classical access control approaches. As DLBAC learns based on metadata, it obviates the need for attribute/role engineering and updates, policy engineering, etc. We also implement $\text{DLBAC}_\alpha$, a prototype of DLBAC, using both real-world and synthetic datasets with varying complexities. We demonstrated $\text{DLBAC}_\alpha$'s effectiveness as well as its generalizability across different datasets. As the core of DLBAC is a neural network, we applied two different state-of-the-art techniques to understand DLBAC decisions in human terms.

# CHAPTER 4: ADMINISTRATION OF MACHINE LEARNING BASED ACCESS CONTROL

This chapter investigates the administration problem of Machine Learning Based Access Control (MLBAC). In particular, we consider the situations where a trained ML could be either from *symbolic* (e.g., RF) or *non-symbolic* (e.g., neural network) types. The symbolic ML methods represent knowledge in the form of logic or a tree that distinguishes them from non-symbolic ones, either statistical or neural [30]. To the best of our knowledge, our proposed method is the first work towards administration in an ML-based access control system. The outcome of this research has been accepted for publication at the following conference:

- Mohammad Nur Nobi, Ram Krishnan, Yufei Huang, and Ravi Sandhu. "Administration of Machine Learning Based Access Control." In European Symposium on Research in Computer Security, ESORICS 2022, Copenhagen, Denmark, 2022.

## 4.1 Introduction and Motivation

Machine Learning (ML) is used in the field of access control for different purposes such as policy mining [74], attribute engineering [7] and role mining [108]. In traditional access control systems such as RBAC [118] and ABAC [64], the access control decision engine decides accesses based on a written policy (or role assignments, in the case of RBAC). In recent years, researchers have proposed utilizing a trained ML model to make access control decisions, possibly supplementing or even eventually replacing rule-based access control systems. We refer to such systems as machine learning based access control (MLBAC) [30, 33, 77, 93, 109, 127]. We thoroughly discuss these methods in Section 2.3.4. These works have demonstrated that, for a given ground truth of an access control state represented in the form of authorization tuples, MLBAC models could capture that access control state with significantly higher accuracy than the traditional access control models such as ABAC. In addition, some works also demonstrate that MLBAC generalizes better than traditional approaches [30, 33, 77, 109]. (Note that generalization is the ability of a model to

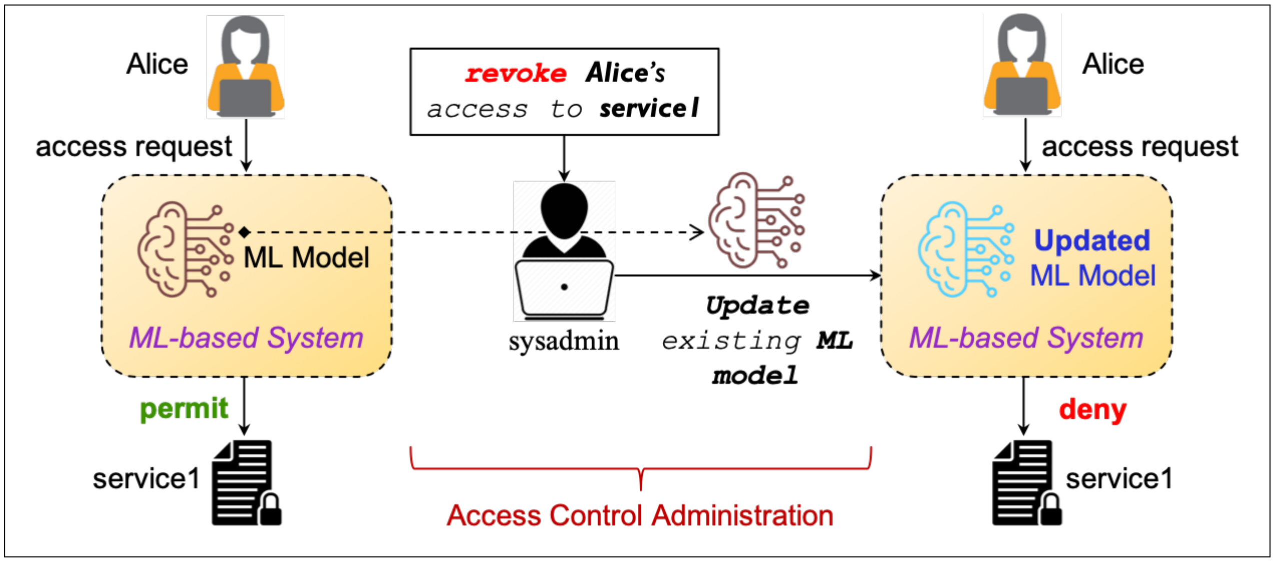

**Figure 4.1**: Administration Problem in ML-based Access Control System.

make accurate decisions on users and resources not explicitly seen during policy mining or ML model training.) Even if MLBAC does not replace traditional forms of access control in practice, it could serve as an effective approach for access control monitoring/auditing or operate in tandem with traditional systems [95, 109, 138].

However, access control systems are not static—changes in access control state are inevitable. A user may be granted new permissions, or some of her current permissions could get revoked. As shown in Figure 4.1, 'Alice' has access to the 'service1' resource. To revoke her access, the learned access control state in MLBAC will need to be correspondingly updated such that it can react accordingly to the applied change. This problem is referred to as *access control administration* in the access control domain [116]. Evidently, administration problems have been thoroughly investigated for traditional approaches [75, 116, 128], but the issue remains entirely unexplored for MLBAC.

Administration challenges could vary from model to model, but the problem's importance remains unchanged. In the case of RBAC, administration activities include assigning/removing permission to/from a role, creating a new role, and managing role hierarchy [116]. For ABAC, administration activities include updating user/resource attributes and policy modification [122]. In such traditional approaches, the changes are accomplished by modifying existing configurations

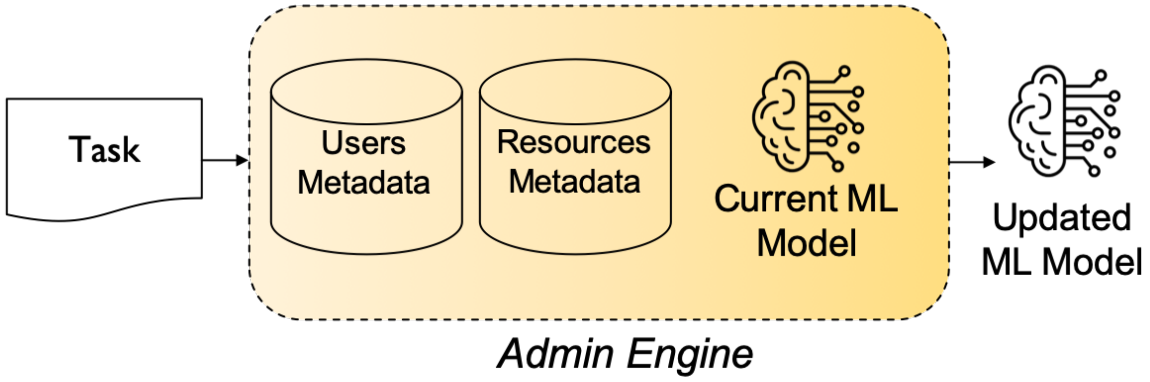

**Figure 4.2**: Overview of MLBAC Administration.

such as written access control policies, and attribute and role assignments. However, in MLBAC, there is no notion of a human-readable written policy to update. If an access control state change is to be made, it requires modification of existing model. Such a modification is complicated as, in most cases, an ML model is a highly complex function, a tree, or even a black-box that a human user can not directly access and modify. Often, to capture changes, one must go through a process similar to the initial training process.

In this chapter, we define MLBAC administration problem and propose a methodology to automate and systematize the MLBAC administration process. Also, we develop two prototypes of administration in a system where access control decisions are made based on either symbolic or non-symbolic ML approaches. Next, we thoroughly evaluate both prototypes for the efficacy of symbolic and non-symbolic ML approaches from an access control administration perspective. Finally, we demonstrate that administration in an MLBAC poses additional challenges and propose different techniques to overcome them.

## 4.2 MLBAC Administration

Figure 4.2 illustrates the overview of MLBAC administration. As depicted in the figure, the *Admin Engine*, the administration framework, takes a change request as the input, which we refer to as a *Task*. We aim to incorporate the requested Task in MLBAC administration. We assume that the Admin Engine has access to the user and resource metadata databases and the ML model, which is currently being used for decision making. We refer to this ML model as *Current ML Model* and denote as $\mathbb{F}_{\mathbf{current}}$. The objective of administration in MLBAC is to modify $\mathbb{F}_{\mathbf{current}}$ to capture the requested Task and generate an updated model. We also refer to this updated model as *Updated ML Model* and designate as $\mathbb{F}_{\mathbf{updated}}$.

### 4.2.1 Requirements

For the purpose of this chapter, we generalize MLBAC as follows. An MLBAC model is trained using the existing access control state of a system, along with various pieces of available metadata values such as those of users and objects. Since the decision engine in MLBAC is an ML model, modifying its access control state is not as obvious as that of, say, ABAC, where a rule is typically adjusted to grant or deny existing accesses. It is often required to modify the model itself to accommodate any authorization-related changes. Consequently, the administrative tasks in MLBAC are somewhat simplified since we no longer need to worry about policy and attribute updates. In this chapter, we focus on basic administrative tasks for MLBAC, including granting/revoking the access of one or multiple users to one or more resources.

Over time, by learning from proposed changes and observing the metadata of users and objects, MLBAC could intelligently adjust other "similar" accesses in the system. We believe smarter access control administration is one of the most significant benefits of MLBAC in practice, hence the focus of this work.

### 4.2.2 Problem Statement and Approach

In an ML model, to modify a piece of learned information, it is required to iteratively update the weights of its neurons (in the case of neural network) or parameters (for classical ML) starting with random initialization [131]. Consequently, we state the MLBAC administration problem as:

*'Given an administrative task, update the MLBAC model's weights and/or parameters such that the updated model captures the desired changes in the access control state.'*

By desired changes, we mean both the given administrative task and additional administrative tasks that are similar to the given task. The challenge specifically, then, is: what is the best approach to accurately learn to accommodate the given task and perform additional changes that are *similar* to the proposed task while keeping the existing access control state unchanged for all other users and resources that are vastly dissimilar? We indicate similar changes as changing access to other users and resources similar to the user and resource given in the proposed task. However, determining whether two users (or resources) are similar or not depends on the *type* of their metadata. If the metadata is real-valued (e.g., age, salary, etc.), it is possible to automatically determine other similar users and resources using distance measurement or clustering approaches [146]. This is also applicable for ordinal categorical values, where there is a notion of *order* among its values (e.g., degrees, clearances, job roles, etc.). However, in the case for nominal categorical values (e.g., department, expertise, etc.), there is no notion of order among them, and therefore one could only perform an equality check based on their values.

In practice, one could anticipate a mix of real-valued, ordinal categorical, and nominal categorical data. The model would automatically find additional similar administrative tasks for real-valued and ordinal categorical metadata values. For nominal categorical metadata values, we seek input from the system administrator (sysadmin) to determine the similarity between the user(resource) involved in the proposed task and other users(resources) in the system. We assume sysadmin will provide some 'similarity measurement criteria' in terms of user and resource metadata, which we refer to as *Criteria*. We further illustrate its syntax in Section 4.2.3.

### 4.2.3 Terminologies

This section introduces some terminologies that we repeatedly use for the explanation of administration in MLBAC.

- **Authorization Tuple.** An Authorization Tuple is user-permissions tuple $\langle user, resource, permissions \rangle$ that specifies the permissions of a $user$ to a $resource$. For example, an Authorization Tuple $\langle$*u1, r1*, {*op1, op3*}$\rangle$ indicates that a user *u1* has operations *op1* and *op3* access to a resource *r1*.

- **Task.** A Task is a change request which is expressed through a tuple of four elements $\langle user, resource, operation, access\rangle$, where the *access* could be either permit or deny. For example, a Task $\langle u1, r1, op3, deny\rangle$ is a request to revoke the *op3* access of the user *u1* to the resource *r1*.

- **AAT (Admin Authorization Tuple).** AAT is an updated Authorization Tuple generated from an existing one based on a Task. For example, $\langle$*u1, r1,* $\boldsymbol{op3, deny}\rangle$ is a given *Task*. Suppose that the existing Authorization Tuple in the system for user *u1* and resource *r1* is $\langle u1, r1, \{op1, \boldsymbol{op3}, op4\}\rangle$. The AAT with respect to the given Task would be $\langle u1, r1, \{op1, op4\}\rangle$. AAT, in effect, is the change that the admin seeks to make.

- **Criteria.** Criteria is defined as a tuple of user metadata name and value pairs and resource metadata name and value pairs that is expressed as:

$$\Big\langle \{umeta_0 \in \{val_0,\, \ldots,\, val_i\},\, \ldots,\, umeta_m \in \{val_0,\, \ldots,\, val_j\}\},$$
$$\{rmeta_0 \in \{val_0,\, \ldots,\, val_k\},\, \ldots,\, rmeta_n \in \{val_0,\, \ldots,\, val_l\}\}\Big\rangle$$

  where $\{umeta_0,\, \ldots,\, umeta_m\}$ is a set of user metadata names, $\{rmeta_0,\, \ldots,$ $rmeta_n\}$ is a set of resource metadata names, and $\{val_0,\, \ldots,\, val_q\}$ indicates possible values for respective metadata name. For example, a sample Criteria could be $\langle umeta1 \in \{val_0,\, val_1\},\, rmeta4 \in \{val_2\}\rangle$. In this Criteria, the possible values of *umeta1* are $val_0$ and $val_1$, and the possible value of *rmeta4* is $val_2$.

- **Additional AAT.** The Additional AAT is a set of *similar* AAT determined based on users and resources similar to the user and resource in the Task. This chapter determines similar users/resources based on the input Criteria. Section 4.2.4 discusses Additional AAT generation.

- **OATs (Other Authorization Tuples).** The OAT is a set of Authorization Tuples in the access control system that excludes AAT and Additional AAT.

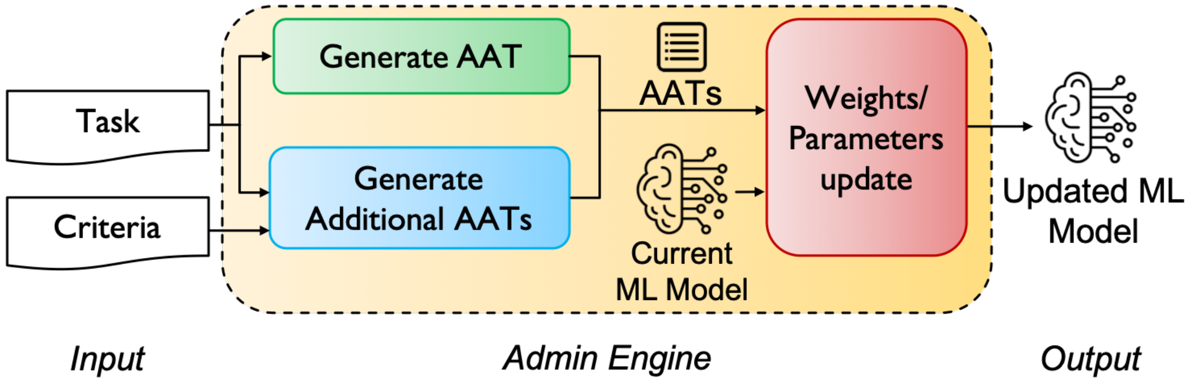

**Figure 4.3**: Administration Process Flow in MLBAC.

### 4.2.4 Methodology

As shown in Figure 4.3, the Admin Engine generates an Admin Authorization Tuple (AAT) for the given Task as described in Section 4.2.3. The AAT is the Authorization Tuple that we aim to integrate into the current ML model, $\mathbb{F}_{\mathbf{current}}$. Next, the Admin Engine generates Additional AAT based on the Task and Criteria. Both AAT and Additional AAT are independent of each other hence it is not required to maintain any specific order for their generation.

We generate Additional AAT based on a set of *similar* users and resources determined using Criteria, as discussed in Section 4.2.2. The Criteria consist of user and resource metadata names and value pairs. For each user, we compare their metadata values with respective metadata values stipulated in the Criteria. If it matches, we call the corresponding user as the *similar user*. Eventually, we determine all the similar users and follow the same approach for finding similar resources. These similar users and resources are the candidate users and resources for the Additional AAT generation. Next, we iterate over the list of candidate users and candidate resources to make user-resource pairs and determine a list of operations for each pair that the user has access to the resource. The user-resource pair with their access operation is an Additional AAT. Eventually, we obtain all the Additional AAT for the given Task and Criteria by repeating the same process. We depict the pseudo-code of Additional AAT generation in Algorithm 4.1. Collectively both AAT and Additional AAT are stored in a set that we refer to as **AATs**. Note that the 'size of AATs' indicates the number of elements in the set.

**Algorithm 4.1** Additional AAT Generation

```
inputs: Task, Criteria
output: Additional AAT
data: uList[ ] and rList[ ] are the list of all users and resources, respectively, with their metadata
      values in the system
generateAdditionalAAT (Task, Criteria)
    additionalAAT[ ] = Ø//An empty list of Additional AAT
    //The extractOperationAccess is a utility function that takes a Task,
      //and returns operation and access from the Task
    operation_t, access_t = extractOperationAccess(Task)
    //getSimilarUsers and getSimilarResources are the utility functions
      //that take list of all users and resources, respectively,
      //and the Criteria, and return similar users and resources
    candidateUsers[ ] = getSimilarUsers(uList, Criteria)
    candidateResources[ ] = getSimilarResources(rList, Criteria)
    //getAccessOperations is a utility function that takes a user and
      //resource, returns user's set of access operations to the resource
    foreach user_c in candidateUsers do
        foreach resource_c in candidateResources do
            Op_c[ ] = getAccessOperations(user_c, resource_c)
            if Op_c $\neq \emptyset$ then
                if (access_t = permit AND operation_t not in Op_c) OR (access_t = deny AND
                  operation_t in Op_c) then
                    cAAT [ ] = Ø
                    cAAT.append(user_c)
                    cAAT.append(resource_c)
                    if access_t = permit then
                        Op_c.append(operation_t)
                    else
                        Op_c.remove(operation_t)
                    end
                    cAAT.append(Op_c)
                    additionalAAT.append(cAAT)
                end
            end
        end
    end
    return additionalAAT
```

At this point, the Admin Engine has the input (AATs) to accommodate in its $\mathbb{F}_{\mathbf{current}}$ model and adjust the weights/ parameters to react accordingly for the newly added changes and produce the $\mathbb{F}_{\mathbf{updated}}$ model. This accommodation is not straightforward, and there are multiple underlying challenges. A naive solution to this problem could be to *retrain* an ML model based on newly generated AATs and original training data that we used to train the $\mathbb{F}_{\mathbf{current}}$. While this approach has its benefits, there are multiple shortcomings. For example, to retrain an ML model, one always has to maintain the original training dataset and the AATs of each administration. Also, retraining is expensive in terms of training time and resource consumption. Therefore, it might not be practical nor feasible for many systems to retrain an ML model to accommodate new changes.

A potential solution could be to update the weights/ parameters of the $\mathbb{F}_{\mathbf{current}}$. However, the process of updating the weights/ parameter values in an ML model has a direct correlation with the type of model in question. Suppose the underlying model is a classical ML algorithm such as SVM and Ensemble Methods. In that case, an incremental machine learning (a.k.a online learning) technique could be a prospective solution [17]. If the model is a neural network, a possible technique could be to use *fine-tuning* [82]. Fine-tuning performs internal adjustments to a trained neural network's (e.g., $\mathbb{F}_{\mathbf{current}}$ in MLBAC) weight based on a set of given examples (AATs in the case of MLBAC).

## 4.3 MLBAC Administration Prototype

This section implements two prototypes of MLBAC administration using ML models from symbolic and non-symbolic classes. We experiment with MLBAC administration to assess how well it reacts to the administrative changes. We apply MLBAC administration in a synthetically generated extensive system with thousands of users and resources. The following sections briefly introduce the simulated system, the ML models, and different administration strategies.

### 4.3.1 System for MLBAC Administration Experimentation

Access control administration is a continuous process where one can expect many change requests during the life of a system. A limited number of real-world access control-related datasets are available from Amazon [10, 11]. These datasets have been used extensively in the literature for ML model training and ABAC policy mining and evaluating how accurately the trained model or mined policy can make access decisions [30, 74, 109]. For any access control administration experiment, we need a system where we will have continuous change requests during the system's life. The Amazon datasets in themselves do not provide such administrative tasks. We generate eight different datasets using the data generation algorithm proposed by Xu et al. [143] and briefly discuss them in Section 3.3.1. Out of eight datasets, we experiment with the administration on the $u5k$-$r5k$-$auth12k$ dataset. The simulated system has around five thousand users and five thousand resources. Also, there are eight user and eight resource metadata values for each user and resource, respectively, and four operations. The dataset contains *nominal* categorical metadata values, as integers, of which each value represents a category. (We briefly discussed nominal and ordinal categorical data in Section 4.2.2.) When visualizing the dataset using t-SNE [135], we found that the samples overlap significantly and are not easily separable, indicating the simulated system is fairly complex [30, 109]. Figure 4.4 illustrates the visualization of $u5k$-$r5k$-$auth12k$ dataset. We train ML models for MLBAC using this dataset.

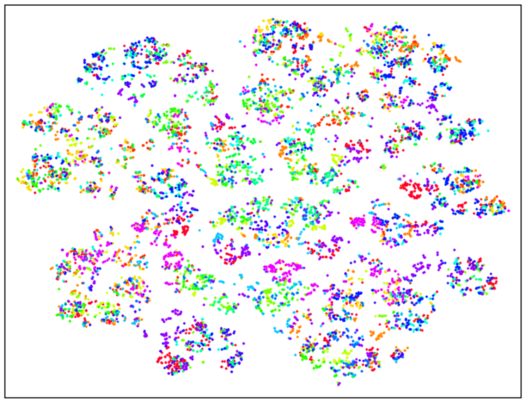

**Figure 4.4**: t-SNE Visualization of the $u5k$-$r5k$-$auth12k$ Dataset.

### 4.3.2 Symbolic and Non-Symbolic ML Models

Among symbolic approaches, the RF algorithm got special attention in the access control domain due to its expressiveness of a decision in the form of a rule [30, 93, 127]. RF can achieve excellent performance in capturing the access control state of a system. However, if the access control state of the underlying system is complicated, a non-symbolic method such as a neural network-based system shows superior performance compared to the symbolic ones [30, 109]. In this work, we develop an MLBAC administration prototype with an RF from the symbolic class to determine its efficiency from an administration perspective, which we refer to as RF-MLBAC. We also investigate another prototype with the neural network from the non-symbolic type. In particular, we consider ResNet [59] as our candidate neural network and refer to it as ResNet-MLBAC. We note that one could use other neural networks, including MLP [120], DenseNet [69], etc., although we do not anticipate any significant changes in our results.

Both RF and ResNet in MLBAC take user/resource metadata values as input to make corresponding access control decisions. Since the metadata values in our dataset are categorical, we encode them before applying them to the model [57]. Our experiment's ResNet architecture has a depth of 8, and the RF has 100 estimators (decision trees in the forest). As the dataset has four different operations, both models output the *probability* of granting the permission for a related operation. Given a feature vector $x$ of the user and resource metadata, the ML model is defined as a prediction function $f$: $\hat{y} = f(x)$, where $\hat{y}$ is the predicted label or permission (grant (1) or deny (0)) of the operation $op$, obtained from comparing the probability of granting the permission from the output of the ML model with a *threshold*. We consider a threshold of 0.5 for our experiment.

### 4.3.3 Administration Strategies in MLBAC

We follow multiple strategies for accommodating a given Task in MLBAC. From a Task perspective, we propose single-Task and multi-Task administration approaches. Also, we examine two learning strategies that include retraining and sequential learning. We discuss them below.

**Single-Task Administration.** We administer each Task individually and replace the current ML model ($\mathbb{F}_{\mathbf{current}}$) with the updated model ($\mathbb{F}_{\mathbf{updated}}$). In this case, we simulate that the sysadmin updates the underlying ML model after receiving any new Task. For single-Task administration, the Admin Engine determines AATs (i.e., both AAT and Additional AAT) for the given Task and then apply the generated AATs for the administration. Figure 4.3 is an illustration of single-Task administration.

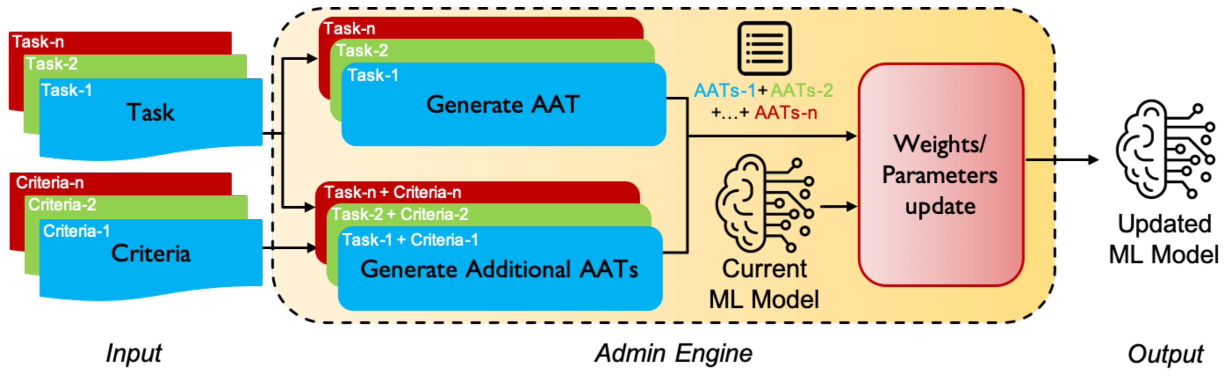

**Figure 4.5**: Multi-Task Administration Process Flow in MLBAC.

**Multi-Task Administration.** In practice, sysadmin may receive multiple unique Tasks together, or they may wait for additional Tasks to accumulate before initiating an administrative operation. To simulate this, we investigate administrations with multiple Tasks simultaneously in MLBAC, which we refer to *multi-Task administration*. In this case, Admin Engine determines AATs for each Task individually and combines them. Then, the combined AATs are used for the administration. We use the term *Task count* to refer to the number of Tasks we consider for a multi-Task administration. Figure 4.5 illustrates multi-Task administration process for n-Tasks. We experiment with 2-Tasks, 3-Tasks, and 6-Tasks administration for multi-Task administration with Task counts 2, 3, and 6.

**Retraining.** A naive solution to the administration problem could be to *retrain* an ML model based on newly generated AATs and initial training data that we used to train the $\mathbb{F}_{\mathbf{current}}$. The idea is to train a fresh model from scratch with the dataset that combines both initial training data and the samples generated for the respective Task (AATs in case of MLBAC)), as shown in Figure 4.6 (left). The trained model will replace the existing $\mathbb{F}_{\mathbf{current}}$ model.

Retraining may not be feasible or practical for many systems due to some reasons. For instance, this process requires storing the *entire* initial training data for future administration. Also, retraining is expensive in terms of computation time and resource consumption. Model training is one of the most time-consuming parts of ML-based applications. One needs to spend the same amount of time to accommodate any new change in an existing system. Besides, retraining an ML model produces a new model that does not hold any previous history of access change and only

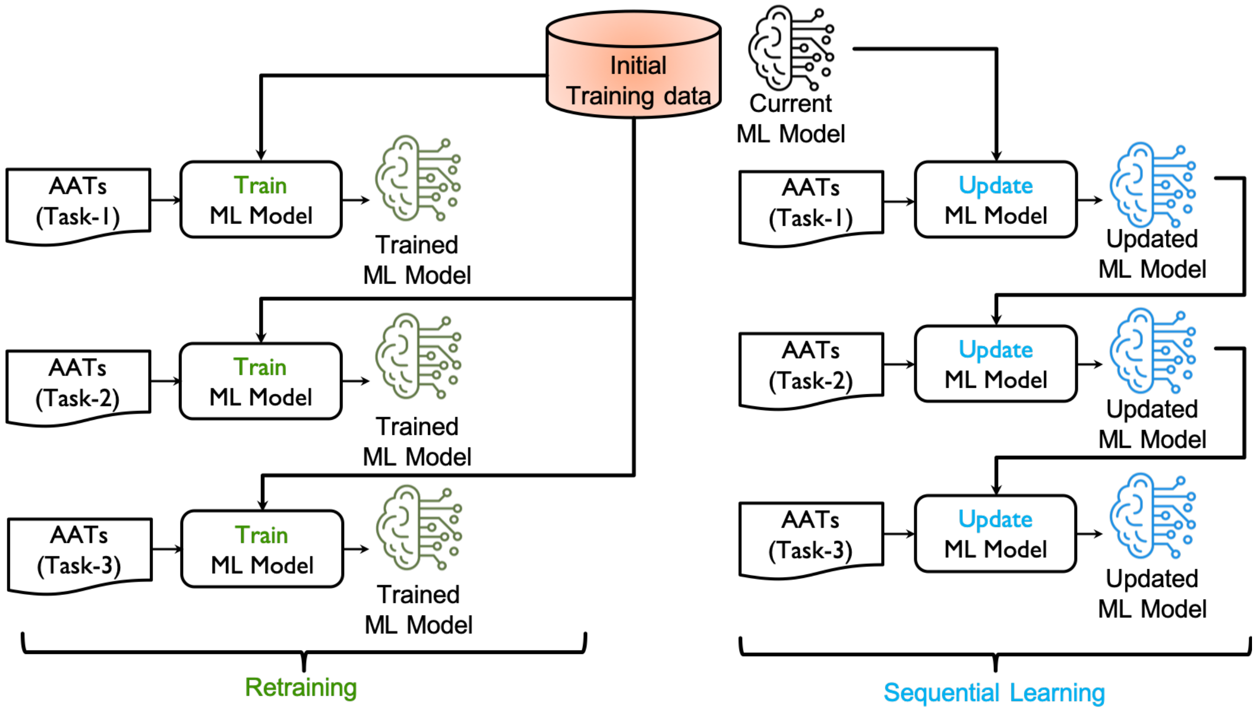

**Figure 4.6**: Retraining (left) vs. Sequential Learning (right) Strategies.

portrays the data provided during retraining.

**Sequential Learning.** Sequentially learning Tasks is vital for developing an autonomous system. It reflects how a human learner identifies the materials to be learned [68, 97]. We also embrace this learning phenomenon to administer MLBAC, as illustrated in Figure 4.6 (right). As shown, for Task-1, the Admin Engine utilizes the $\mathbb{F}_{\mathbf{current}}$ and AATs of Task-1 to update the existing model and generate $\mathbb{F}_{\mathbf{updated}}$ model. This $\mathbb{F}_{\mathbf{updated}}$ model replaces the $\mathbb{F}_{\mathbf{current}}$ model for deciding accesses and acts as input for the Task-2 administration.

### Sequential learning process and its effect

For sequential administration in RF, we append additional estimators that learn new changes while keeping existing estimators untouched. Even though this method is not as efficient as incremental learning in other classical approaches, we did not find any better strategy for RF in the literature. For each single-Task administration, we append two estimators in the RF model. We append 5, 8, and 10 estimators for 2-Tasks, 3-Tasks, and 6-Tasks administration, respectively. Note that we performed trial and error with more estimators; however, increasing this number of estimators was efficient in performance and model size. On the other hand, for ResNet, we employ the *fine-tuning* technique (discussed in Section 4.2.4) to incrementally learn new changes and update the network's weights accordingly.

However, one of the major challenges of sequential learning is maintaining the existing access control state unchanged. An ML model could *forget* previous knowledge while learning new information. For example, Alice has read and write access to a resource projectA, and Bob has only read access to another resource projectB. Sysadmin received a Task '*to permit Bob with the write access to projectB*'. After administering the requested Task, the system correctly updates the access control state such that Bob has both read and write access to the projectB. However, there might be a case that the updated system could not make the correct access decision for Alice to the projectA. In other words, the system *forgot* Alice's access to projectA. Formally, in ML arena, this phenomenon is known as *catastrophic forgetting* [84, 144]. In the case of RF, this is not a problem as the technique we followed does not modify existing estimators in the model but append new ones. However, this is a significant challenge for neural networks since the knowledge of the previously learned Task(s) starts decaying with the incorporation of the new Task [84].

**Overcoming Catastrophic Forgetting in MLBAC.** Catastrophic forgetting is a well-known problem in machine learning while updating the model, especially when dealing with a neural network. Fortunately, this is a well-studied problem in ML literature, and different approaches have been proposed to overcome this hurdle [68, 123, 144]. One of the common strategies is to replay previous knowledge in the form of training data (the dataset used to train the network) with new samples (AATs in MLBAC) during fine-tuning [123]. It may not be practical to store the training data in many applications if the samples are too large (e.g., image, video, etc.). However, this is not an issue for MLBAC as it works based on numerical user and resource metadata and attributes values. In our simulated system, each training sample is a *vector* of user and resource numerical metadata. Other real-world datasets reported in [10] and [11] are also similar, which indicates the feasibility of storing the prior training data for MLBAC. To minimize the required storing space, we reserve

**Table 4.1**: List of Task and Criteria.

| Task Id | Task | Criteria | Size of AATs |
|---|---|---|---|
| t-1 | $\langle uid = 259, rid = 112, op3, permit\rangle$ | $\langle umeta0 \in \{9\}, umeta6 \in \{6\}, rmeta0 \in \{9\}, rmeta3 \in \{46\}\rangle$ | 43 |
| t-2 | $\langle uid = 4624, rid = 4634, op4, deny\rangle$ | $\langle umeta2 \in \{58, 49\}, umeta3 \in \{39\}, rmeta3 \in \{39\}\rangle$ | 94 |
| t-3 | $\langle uid = 1992, rid = 1858, op1, permit\rangle$ | $\langle umeta2 \in \{11\}, rmeta2 \in \{11\}, rmeta3 \in \{48, 91\}\rangle$ | 92 |
| t-4 | $\langle uid = 5049, rid = 5177, op4, permit\rangle$ | $\langle umeta1 \in \{6\}, umeta4 \in \{47, 71\}, rmeta1 \in \{6\}\rangle$ | 215 |
| t-5 | $\langle uid = 2034, rid = 2041, op2, deny\rangle$ | $\langle umeta4 \in \{10\}, rmeta1 \in \{6, 10\}, rmeta4 \in \{10\}\rangle$ | 75 |
| t-6 | $\langle uid = 1348, rid = 1083, op2, permit\rangle$ | $\langle umeta3 \in \{46, 50, 53\}, umeta5 \in \{13\}, rmeta3 \in \{46, 50, 53\}, rmeta5 \in \{13\}\rangle$ | 187 |
| t-7 | $\langle uid = 1345, rid = 1092, op4, permit\rangle$ | $\langle umeta0 \in \{24, 64\}, umeta6 \in \{7\}, rmeta0 \in \{24, 64\}, rmeta6 \in \{7\}\rangle$ | 139 |
| t-8 | $\langle uid = 442, rid = 580, op3, permit\rangle$ | $\langle umeta3 \in \{49\}, umeta5 \in \{47, 111\}, rmeta5 \in \{47, 111\}, rmeta7 \in \{49\}\rangle$ | 134 |
| t-9 | $\langle uid = 2599, rid = 2593, op1, permit\rangle$ | $\langle umeta0 \in \{11\}, umeta1 \in \{17\}, rmeta0 \in \{11\}, rmeta1 \in \{17\}\rangle$ | 66 |
| t-10 | $\langle uid = 4112, rid = 1241, op2, permit\rangle$ | $\langle umeta1 \in \{18\}, rmeta1 \in \{18\}, rmeta3 \in \{45, 47, 113\}\rangle$ | 75 |
| t-11 | $\langle uid = 2135, rid = 4875, op3, deny\rangle$ | $\langle umeta2 \in \{13\}, umeta4 \in \{71, 96\}, rmeta2 \in \{13\}, rmeta4 \in \{71, 96\}\rangle$ | 118 |
| t-12 | $\langle uid = 660, rid = 560, op1, permit\rangle$ | $\langle umeta3 \in \{88\}, umeta5 \in \{48, 111\}, rmeta5 \in \{48, 111\}, rmeta7 \in \{88\}\rangle$ | 107 |
| t-13 | $\langle uid = 2019, rid = 2056, op2, deny\rangle$ | $\langle umeta4 \in \{12\}, rmeta1 \in \{78, 82\}, rmeta4 \in \{12\}\rangle$ | 121 |
| t-14 | $\langle uid = 1228, rid = 1088, op1, permit\rangle$ | $\langle umeta2 \in \{11, 63\}, umeta5 \in \{20\}, rmeta5 \in \{20\}\rangle$ | 97 |
| t-15 | $\langle uid = 2825, rid = 3044, op2, permit\rangle$ | $\langle umeta6 \in \{8\}, rmeta1 \notin \{6, 10\}, rmeta2 \in \{61, 62\}, rmeta6 \in \{8\}\rangle$ | 107 |
| t-16 | $\langle uid = 965, rid = 861, op4, permit\rangle$ | $\langle umeta3 \in \{45\}, umeta7 \in \{20\}, rmeta3 \in \{45\}, rmeta6 \in \{20\}\rangle$ | 63 |
| t-17 | $\langle uid = 3745, rid = 3843, op3, permit\rangle$ | $\langle umeta0 \in \{31\}, umeta6 \in \{2, 5, 9, 18\}, umeta7 \in \{4, 13\}, rmeta0 \in \{31\}\rangle$ | 83 |
| t-18 | $\langle uid = 2488, rid = 2495, op3, permit\rangle$ | $\langle umeta1 \in \{58\}, rmeta1 \in \{58\}, rmeta2 \in \{58, 61\}\rangle$ | 116 |

a quarter of all the original training samples instead of keeping the entire training dataset, which we refer to as *Replay Data*. The Admin Engine can access the Replay Data and add them during administration along with AATs for the correspondent Task. After performing an administration, we append a quarter of AATs to the Replay Data, which reflects the information of the current Task during the next administration. Admin Engine always ensures the AATs and Replay Data are mutually exclusive.

## 4.4 Evaluation

This section measures the performance for our simulated system to determine the feasibility and efficacy of the proposed strategies for MLBAC administration.

### 4.4.1 Evaluation Methodology

We experiment and evaluate administration performance in RF-MLBAC and ResNet-MLBAC prototypes on $u5k$-$r5k$-$auth12k$ dataset. The dataset has around twelve thousand samples (Authorization Tuples). We use 80% of the samples for training and 20% for testing the ML models. After initial training, the trained models' (RF and ResNet) performances are above 99% for the test samples, respectively, implying that $\mathbb{F}_{\mathbf{current}}$ is highly accurate in making access decisions.

We create a set of Tasks to simulate that the sysadmin received all those Tasks one by one during the system's life span. To that extent, we construct eighteen (considered randomly) distinct Tasks from the $u5k$-$r5k$-$auth12k$ dataset with different kinds of Criteria. Table 4.1 reports all the simulated Tasks and respective Criteria. For example, the Task Id *t-1* indicates the first Task and a Task: $\langle uid = 259, rid = 112, op3, permit\rangle$ means a user with *uid*=259 needs *op3* access to a resource with *rid*=112. Also, the Criteria: $\langle umeta0 \in \{9\}, umeta6 \in \{6\}, rmeta0 \in \{9\}, rmeta3 \in \{46\}\rangle$ specifies the user whose *umeta0* and *umeta6* metadata values are 6 and 9, respectively, could have *op3* access to resources with *rmeta0* and *rmeta3* metadata values 9 and 46, respectively. Besides, based on the Task and Criteria, number of generated Additional AAT is 42 and combining the AAT gives a AATs of size 43. We ensure that every Task is independent concerning its change request and purpose. While updating the model, we use 80% of the AATs for training and 20% for testing the updated ML model. We have created a repository on GitHub consisting of the source code, dataset, and respective AATs, OATs, and ReplayData for each Task[1].

We evaluate the administration performance in terms of accuracy. We define the *accuracy* as the measure (in percentage) of correct access authorization for a user to a resource with respect to the actual access control state (ground truth).

### 4.4.2 Results

To evaluate the administration performance in RF-MLBAC and ResNet-MLBAC, we assess how accurately the $\mathbb{F}_{\mathbf{updated}}$ model can capture the AATs (both AAT and Additional AAT). We also evaluate how well it can preserve the access control state of all other users and resources (OATs as described in Section 4.2.3).

We experiment and evaluate the performance for all eighteen Tasks with single-Task and multi-Task administrations. For multi-Task administration, we consider 2-Tasks, 3-Tasks, and 6-Tasks. For example, a 3-Tasks administration indicates, we use three different Tasks together for an administration. In the single-Task administration, it requires *eighteen* different administration to

[1]`https://github.com/dlbac/MLBAC-Administration`

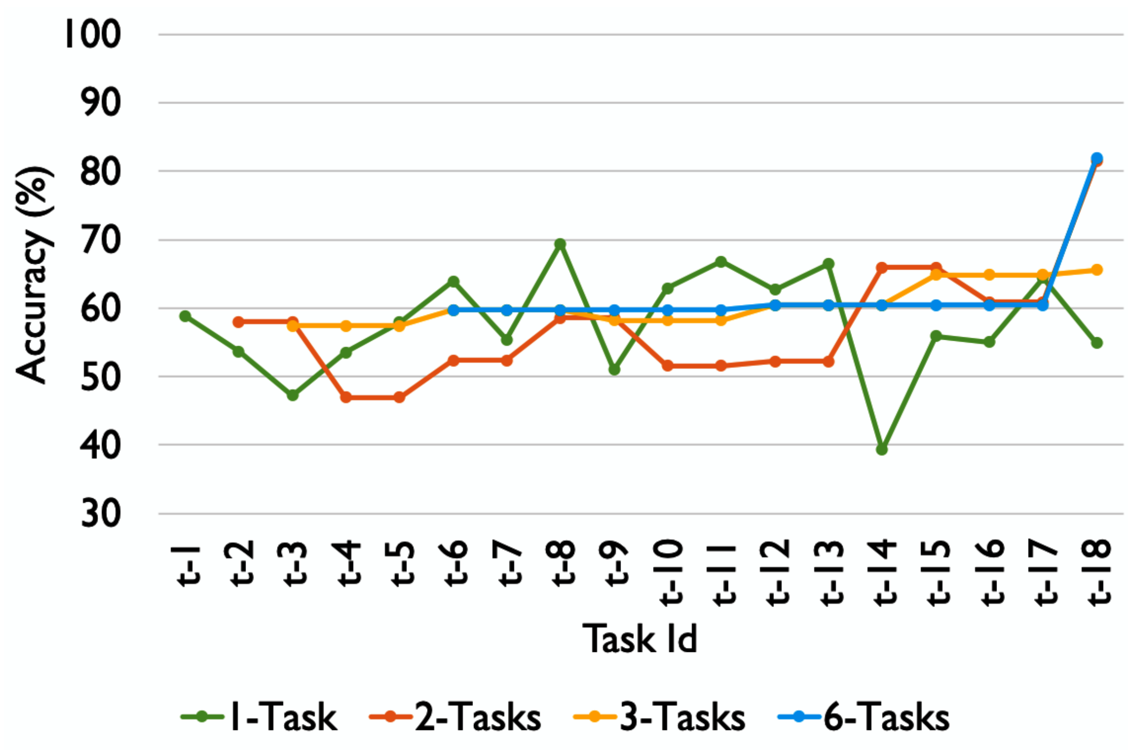

**Figure 4.7**: Administration performance in RF-MLBAC: Retraining

accomplish all the 18 Tasks as it administer one Task at a time. For multi-Task administration, the number of administration reduces with an increase in Task count. For example, for 3-Tasks administration, it requires six different multi-Task administrations to finish all the eighteen Tasks.

**Administration performance in RF-MLBAC**

**Retraining.** As discussed, retraining is a naive approach and inefficient for both data and computation. We evaluate the retraining performance in the $\mathbb{F}_{\mathbf{updated}}$ model. We assess both AATs and OATs performance for single-Task and multi-Task (2, 3, and 6 Tasks) administrations. For OATs, the $\mathbb{F}_{\mathbf{updated}}$ model is as accurate as of the initial trained model with more than 99% accuracy. This performance is consistent across all the single and multi-Tasks administrations. However, in the case of AATs, as demonstrated in Figure 4.7, the accuracy range is 40% to 80%. In almost all the circumstances, the accuracy is inconsistent except for six-Tasks administration that shows a better and more persistent result across administrations with 60% accuracy. Overall, the low AATs performance indicates that the RF model can capture only a portion of new changes while

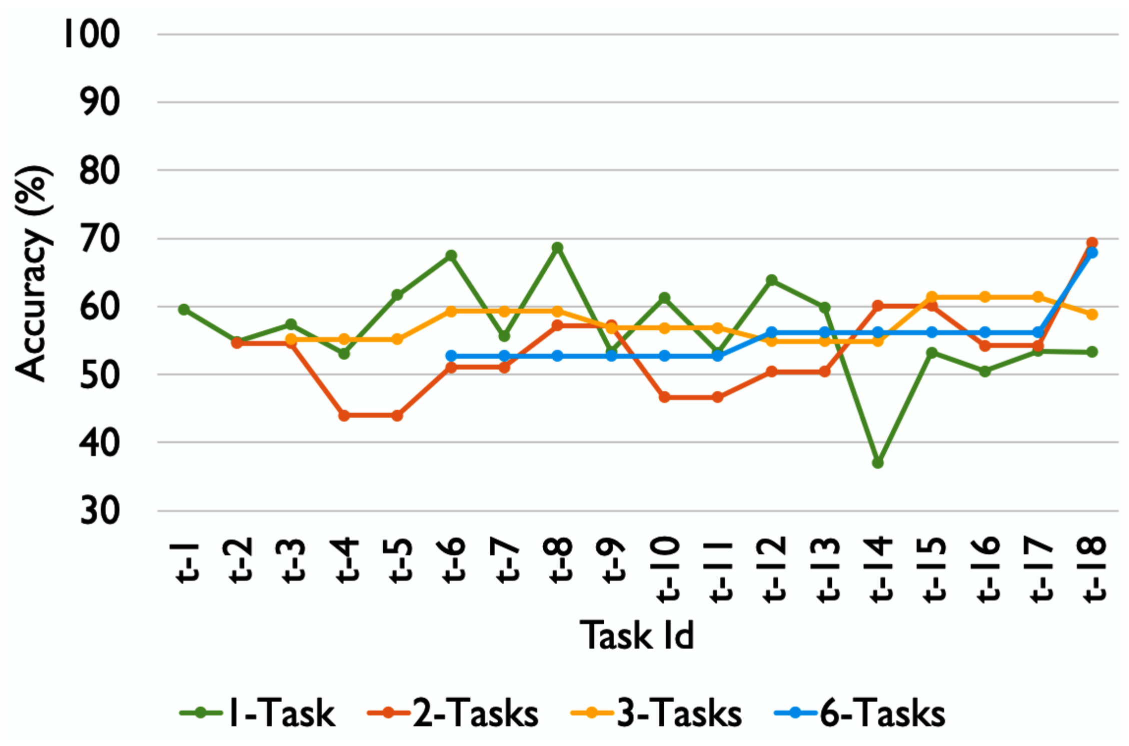

**Figure 4.8**: Administration performance in RF-MLBAC: Sequential Learning

accommodating the proposed Task.

**Sequential Learning.** We perform the same evaluation for sequential learning and apply both single and multi-Tasks administrations. Similar to retraining, the OATs performance is over 99% across administrations and consistent, indicating that the RF model could preserve the existing access control state better. This excellent result indicates that the RF model did not forget the initial access control state. This result is anticipated because RF-MLBAC appends new estimators to comprehend proposed changes instead of modifying their existing estimators while learning new changes.

However, we see a very opposite scenario in AATs performance. As shown in Figure 4.8, the accuracy of AATs varies in the range of 40% to 70%. For single-Task and two-Tasks administrations (green and orange lines in the figure), the AATs performance seems highly inconsistent compared to what we see for six-Tasks administration (blue line). However, the accuracy of six-Tasks administration is about 55%, which indicates that considering many Tasks together for a single administration may not provide a better result. On the contrary, the three-Tasks administra-

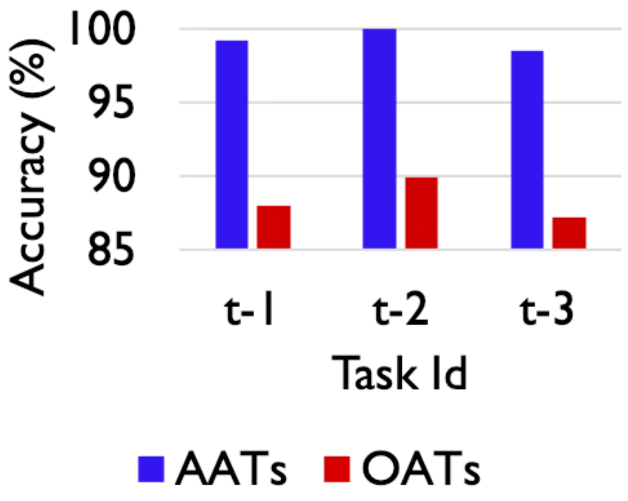

**Figure 4.9**: Administration Performance in ResNet-MLBAC: Without Replay Data

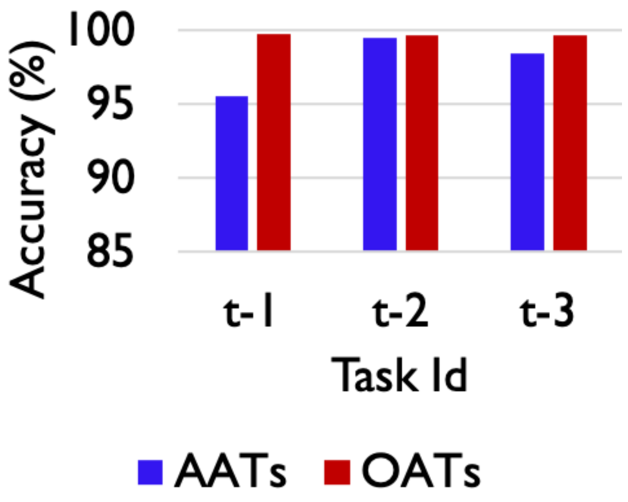

**Figure 4.10**: Administration Performance in ResNet-MLBAC: With Replay Data

tion shows a consistent performance with around 60% accuracy across Tasks, suggesting an RF model's multi-Tasks administration with around three Tasks could be a potential administration to consider.

### Administration performance in ResNet-MLBAC

**Sequential Learning.** Training a neural network from scratch is computationally costly, which is neither efficient nor feasible in access control. As a result, we did not take the naive (retraining) approach for this prototype. To begin with administration in ResNet-MLBAC, we administer the first three Tasks (t-1 to t-3) as a single-Task administration without providing any Replay Data to see the impact of sequential learning in the existing access control state. As shown in Figure 4.9, the updated network captures the authorization for AATs with excellent accuracy, as opposed to what we observed in RF-MLBAC. However, for OATs, we see a lower accuracy (below 90%) in all three cases, indicating the network forgot (catastrophic forgetting) the access control state of a good amount of existing users and resources while learning new information. To overcome that, we apply Replay Data (as discussed in Section 4.3.3) along with AATs during administration. Figure 4.10 illustrates that combining Replay Data with AATs helps ResNet-MLBAC administration to significantly reduce the catastrophic forgetting. As shown in the figure, the accuracy of OATs is now above 99% across all three Tasks. Such a significant increase in OAT performance implies the remarkable impact of Replay Data in overcoming catastrophic forgetting.

Further, we experiment with all eighteen Tasks for single and multi-Task administrations. Figure 4.11 and Figure 4.12 demonstrate the performance of AATs and OATs, respectively. As illustrated, the AATs performance range is 96% to 99% accuracy, significantly better than what we observed in the RF-MLBAC. As we see in the figures, for both AATs and OATs, the performance of 6-Tasks administration is low compared to other multi-Task administrations. Such inferior results indicate the infeasibility of using many Tasks under a multi-Task administration in ResNet-MLBAC administration. Similarly, the performance of AATs in single-Task administration is inconsistent across all the Tasks, which signifies that using single-Task administration in ResNet-MLBAC may produce very unstable performance. However, for both 2-Tasks and 3-Tasks multi-Task administrations, the result is persistent and progressing for both AATs and OATs, thereby proving feasible and better for ResNet-MLBAC administration.

**Summary of Performance in RF-MLBAC and ResNet-MLBAC.** Based on AATs and OATs performance in both prototypes, it is evident that the RF-MLBAC shows better results in preserving the existing access control state. It was expected as we did not change the estimators of the initial RF model. However, the AATs performance in RF-MLBAC is extremely low, indicating it

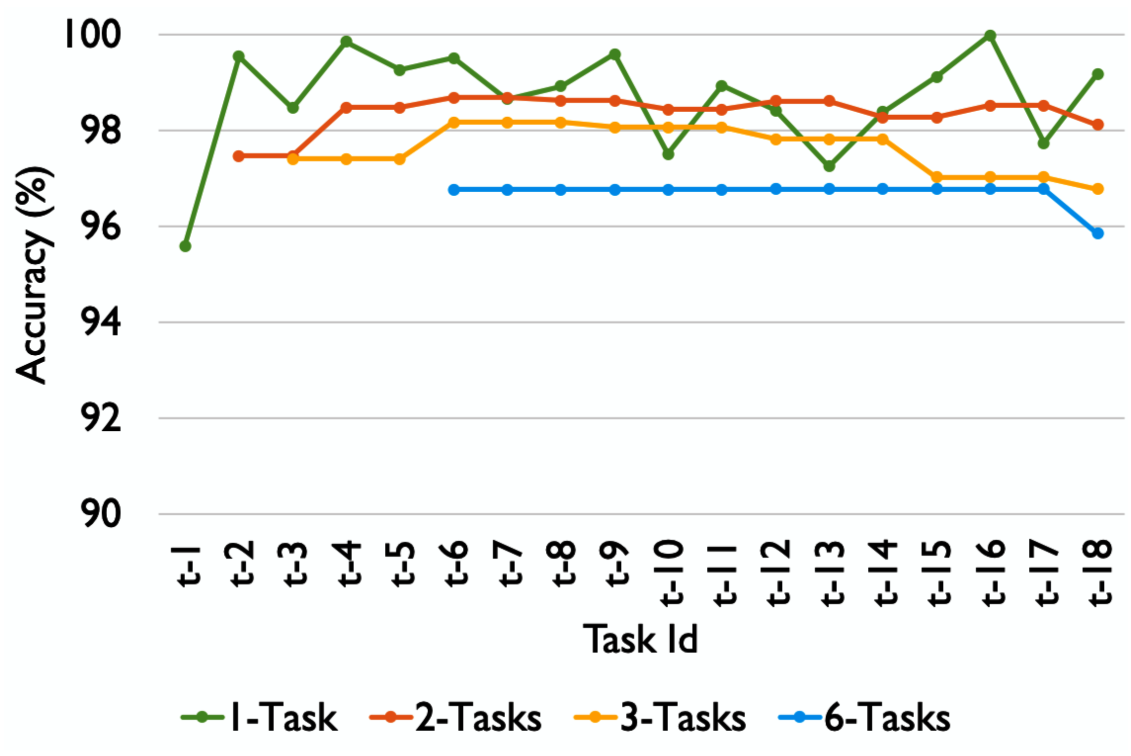

**Figure 4.11**: AATs Performance

could not well capture proposed changes. Besides, it has other drawbacks from model size and optimization perspectives. The size of the trained model gradually grows with the addition of new estimators in it. Also, each administration needs trial and error to determine an optimal number of new estimators to append that correlates with how many Task one consider for a multi-Task administration. On the other hand, ResNet-MLBAC administration shows a better accuracy for AATs and comparable performance in OATs, indicating it could better capture proposed changes while retaining the existing access control state intact.

### 4.4.3 Administration cost evaluation in RF-MLBAC and ResNet-MLBAC

We evaluate administration costs for both administration prototypes concerning computation time for an administration. We observe that RF-MLBAC administration generally takes less than a second to complete an administration (considering a maximum of 18 Tasks for an administration). On the contrary, ResNet-MLBAC administration is slower than RF-MLBAC and varies with the increase of Tasks count for each administration, as depicted in Figure 4.13. We measure the com-

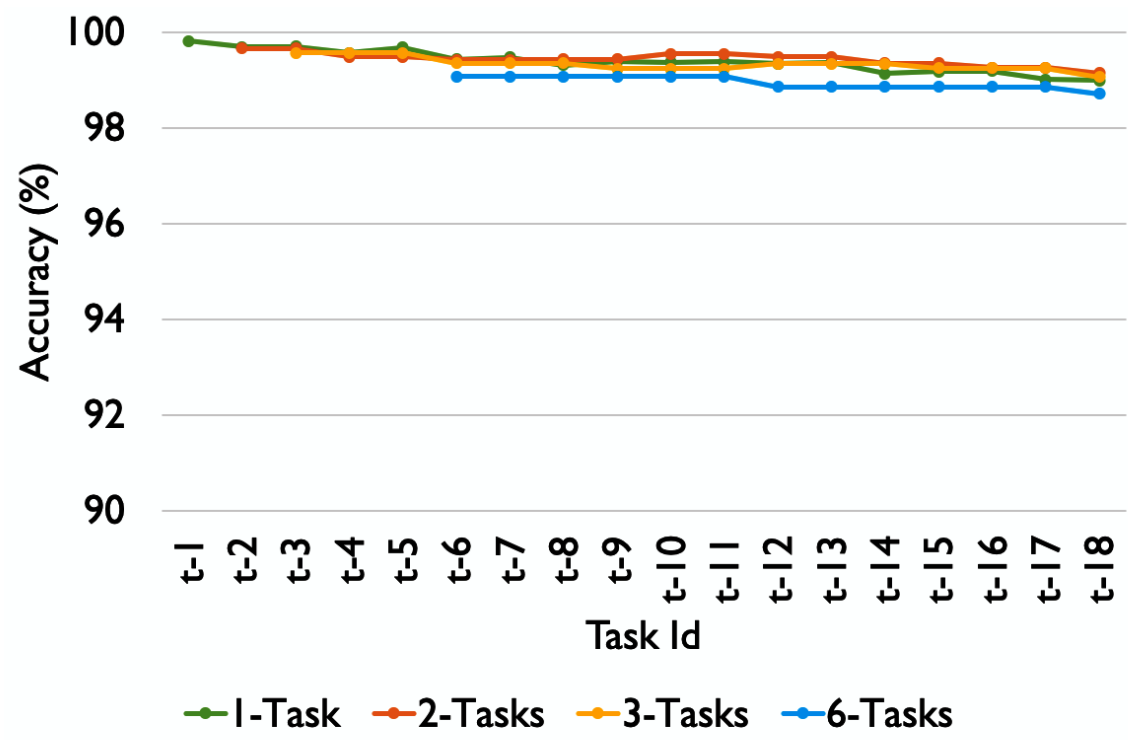

**Figure 4.12**: OATs Performance

putation cost of single-Task and multi-Task administrations in ResNet-MLBAC to identify how administration time varies with the increase in *Task count* and how many Tasks are feasible for a single administration. We consider different sizes (1, 3, 6, 9, 12, 15, and 18 Tasks) multi-Task administration.

As shown in Figure 4.13, it needs 10 seconds for a single-Task administration, which is the same for 3, 6, and 9-Tasks administrations, and becomes double for 12, 15, and 18-Tasks administrations. From a performance perspective, for 3 and 6-Tasks administrations, we see a balanced accuracy in AATs and OATs. Such results suggest the feasibility of using multi-Task administration in ResNet-MLBAC. However, considering many Tasks together (e.g., 12 or 15-Tasks administration) decreases the OATs performance, justifying the infeasibility of administering too many Tasks together under a single administration.

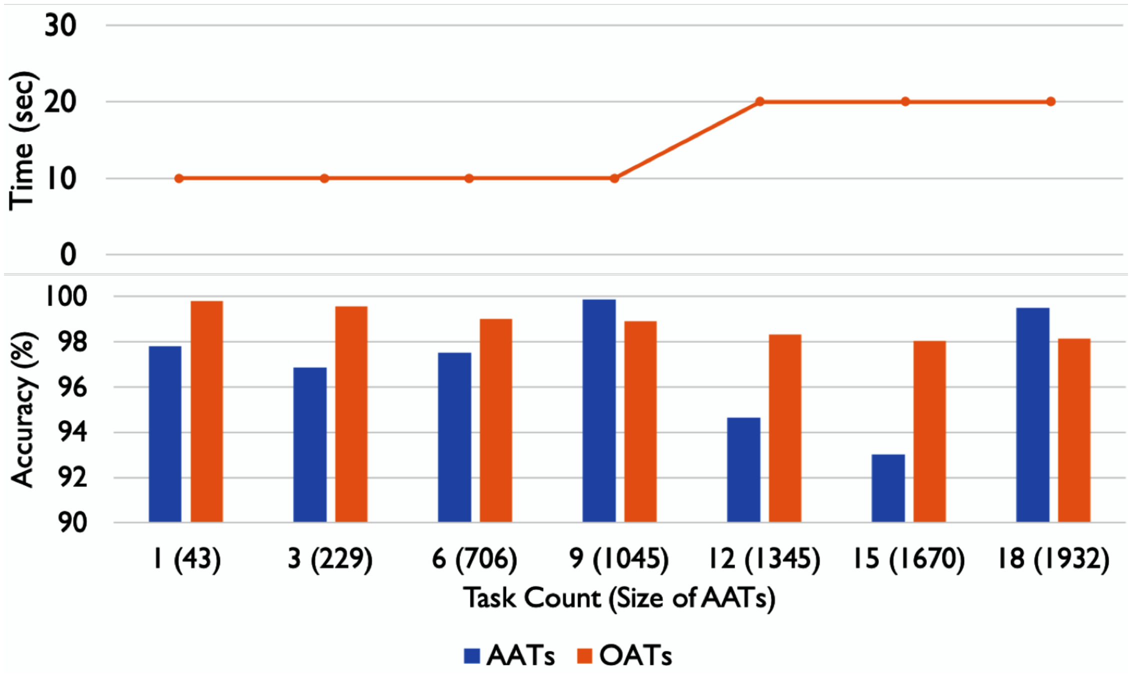

**Figure 4.13**: Administration Cost in ResNet-MLBAC for Sequential Learning.

## 4.5 Conclusion

This chapter explores the access control administration problem for MLBAC. We review administration requirements in the same context and propose an administrative framework for MLBAC administration. We implement two prototypes, RF-MLBAC and ResNet-MLBAC, on a simulated system applying ML algorithms from symbolic and non-symbolic classes. Due to the uniqueness of the MLBAC administration problem, there are many underlying challenges, such as insufficient learning of the proposed changes and forgetting existing access information. We propose different strategies to overcome them. Also, we thoroughly evaluate both prototypes. Our empirical results summarize that the non-symbolic approach performs better than the symbolic one while adjusting for new changes in MLBAC administration.

# CHAPTER 5: ADVERSARIAL ATTACKS IN MACHINE LEARNING BASED ACCESS CONTROL

This chapter investigates adversarial attacks in MLBAC. In particular, we consider an MLBAC where the ML model is a deep neural network. The outcome of this research has been accepted for publication at the following conference:

- Mohammad Nur Nobi, Ram Krishnan, and Ravi Sandhu. "Adversarial Attacks in Machine Learning Based Access Control." In the $1^{st}$ Italian Conference on Big Data and Data Science (ITADATA), Milan, Italy, 2022.

## 5.1 Introduction and Motivation

With advances in Big data, the Internet of Things (IoT), cloud computing, etc., the demand for a more dynamic and efficient access control system is escalating. The traditional systems, such as RBAC [118], ABAC [64], ReBAC [48], etc., have demonstrated their effectiveness in the access control arena for a long time. However, with the increased complexity of computing systems, their generality, flexibility, and usability come at a higher cost and are somewhat inadequate [15, 30, 77]. To deal with such large-scale access control systems, the use of Machine Learning (ML) is becoming common in different areas such as policy mining [37, 74, 76], attribute engineering [7], role mining [108], etc. Contemporary researches also manifest the advantages of using an ML model for more accurate access control decision-making [30, 33, 77, 93, 109, 127]. These systems decide accesses based on a trained ML model instead of a written access control policy. We refer to such systems as machine learning based access control (MLBAC). In MLBAC, the access control decisions (grant or deny) are made using user and resource metadata and attributes. Metadata and attributes are the user/resource features that an ML model learns for subsequent access decisions in MLBAC. For simplicity, we refer to both metadata and attributes as ‘metadata’. These metadata could be expressed in categorical (e.g., ‘security_level,’ ‘designation’) and continuous (e.g., ‘age,’ ‘salary’) data.

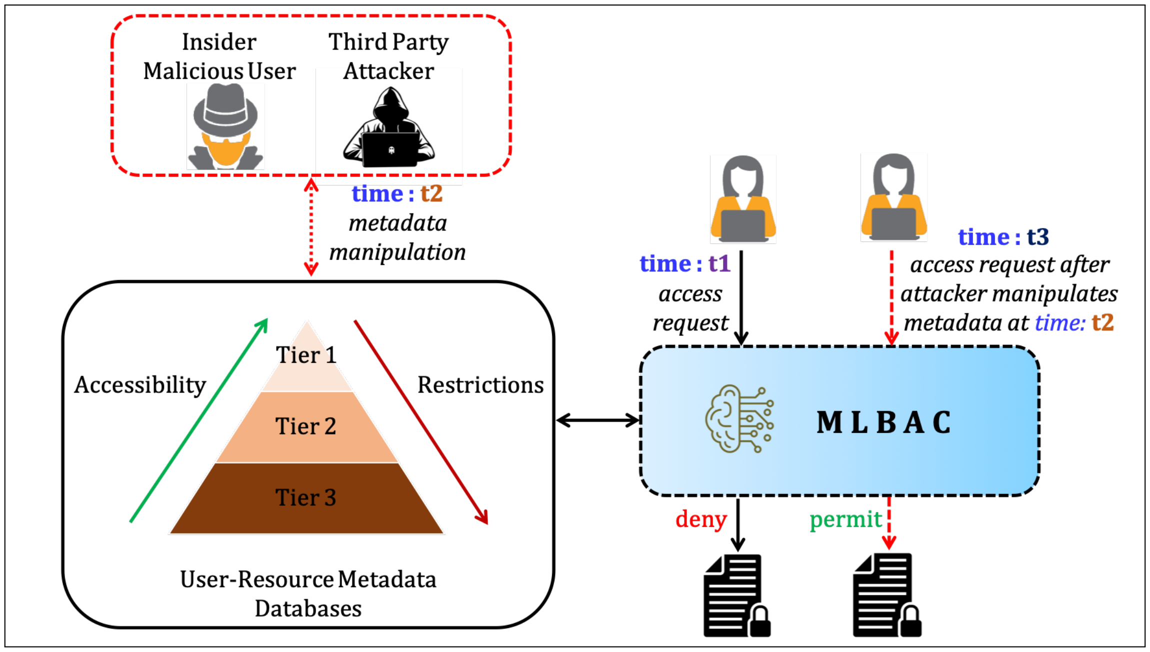

**Figure 5.1**: The Adversarial Attack Problem in MLBAC.

Among different ML models, neural networks are prevalent for obtaining a generalized and accurate MLBAC system due to their ability to capture features from complex input [30, 77, 109]. Such quality of the neural networks poses some security concerns as they are highly sensitive to the minor changes in the input— that is, a slight manipulation or introduction of additional information to the input may result in an unintended output [14]. In ML, this issue is known as an *adversarial attack*, and the manipulated data is an *adversarial example*. For instance, in MLBAC, an attacker could perturb the user/resource metadata to gain access to a resource forcibly. Figure 5.1 demonstrates the adversarial attack problem in MLBAC. An MLBAC is deployed in a system where a user requests to access a resource at time 't1', which the system denies. At time 't2', the user by itself or a third-party adversary purposefully manipulates the respective user and resource metadata to gain access. As illustrated in the figure, system administrators (sysadmin) may store metadata in the databases with different levels of security restrictions, e.g., Tier 1, Tier 2, etc. Therefore, the adversary may or may not equally access and modify each metadata. For example, an adversary may not have access to the 'job_role' metadata as a more restricted database could store them. On the contrary, the adversary may manipulate (e.g., 'expertise' metadata) or influence (e.g., login_frequency) some metadata to which the user may have direct or indirect access. As shown in the figure, after the manipulation, the user requests the same access at the time ‘t3’, which is eventually granted by the MLBAC system.

This area is studied carefully and comprehensively in the domains where inputs are generally images [53, 141], e.g., computer vision. However, this problem is equally important in the access control domain, especially for MLBAC, where the input data is *non-image*. There are a few works where adversarial have been studied where input is a categorical tabular data. For example, Ballet et al. [14] propose an imperceptible adversarial attack for the tabular data domain. The authors manipulate less important features to a greater extent, adding a minimal perturbation to the more important features to ensure imperceptibility. Also, Cartella et al. [32] present an adversarial attack for imbalanced tabular data for fraud detection. The proposed method obtains adversarial examples that are less perceptible when analyzed by humans. Recently, Kumar et al. [87] propose an adversarial attack for a payment system, focusing on generating adversarial examples on the tabular dataset with limited resources (the least number of queries used). The authors experiment with a gradient-free approach in black-box settings. Mathov et al. [96] also propose a framework for adversarial examples in heterogeneous tabular data, including discrete, real-number values, categorical, etc. The proposed framework defines a distribution-aware constraint and then incorporates them by embedding the heterogeneous input into a continuous latent space.

As discussed, in MLBAC, the input is the user/resource’s metadata which is *tabular* data expressed in terms of a set of both categorical and continuous values. Also, in access control domain, metadata may have different levels of restrictions, as shown in Figure 5.1. Therefore, the adversarial attack needs to be investigated concerning the access control domain. In this chapter, we define adversarial attack problems in the MLBAC context. We also propose to customize objective functions imposing additional accessibility constraints. Also, we simulate the adversarial attack cases in the deep neural network-based MLBAC with two complex and large-scale systems. Also, we consider underlying user/resource metadata in both systems contain a mix of categorical and continuous data. Finally, we evaluate the performance of our experimentation and demonstrate that applying accessibility constraints makes it harder for the adversarial attack generation.

## 5.2 Adversarial Attack in MLBAC

### 5.2.1 Adversarial Attack Problem

This section discusses the MLBAC adversarial attack problem. For any access request to the MLBAC, the input to the model is both user and resource metadata, and the MLBAC decides whether the user has access or not. Given the ground truth of an access decision of a request and respective user/resource metadata, we can express them as a tuple of user metadata, resource metadata, and an access decision. We refer to such tuple as an *authorization tuple*. For example, an authorization tuple $\langle$u1, r1, {grant}$\rangle$ indicates that a user u1 has access to the resource r1. The u1 and r1 also represent the set of their respective metadata and metadata values.

We consider $\mathbb{X}$ represents a set of authorization tuples where user and resource metadata are denoted by $x_a$ with $a \in [\,1...N\,]$ and associated with a binary access decision $y_a$ representing either grant (1) or deny (0). Also, $x_a$ is defined by a 'vector' of user and resource metadata values and expressed as: $x_a = \Big\langle m_1^u : v_1\,,\, m_2^u : v_2\,,\, \ldots,\, m_i^u : v_i\,,\, m_1^r : v_1\,,\, m_2^r : v_2\,,\, \ldots,\, m_j^r : v_j \Big\rangle$. The pair $m_i^u : v_i$ indicates $v_i$ is the value of $i^{th}$ user metadata $m_i^u$. Similarly, the pair $m_j^r : v_j$ indicates $v_j$ is the value of $j^{th}$ resource metadata $m_j^r$.

Also, we consider $f : \mathbb{X} \longrightarrow \{\,0, 1\,\}$, a binary classifier mapping to access decisions where the grant is represented by '1' and deny as '0', which is obtained from comparing the probability of access at the output of the classifier with a *threshold*.

For a given user-resource metadata values $x$, the access decision $y = f(x)$, and target access decision $t \neq y$, we aim to generate an optimal perturbation $x_p$ such that

$$f(x) = y \neq f(x + x_p) = t$$

Our goal is to generate an adversarial example such that we can minimize the amount of perturbation ($x_p$) and gain the desired access $t$. To accomplish that, we define our objective function as the accumulation of access change constraint and minimization of $x_p$:

$$g(x_p) \;=\; \mathcal{L}(x + x_p,\, t) \;+\; \omega \parallel x_p \parallel$$

Where $\mathcal{L}(x,t)$ is the value of loss of the binary classifier $f$ calculated for the input $x$ and target access decision $t$. Also, $\parallel x_p \parallel$ measures the $l_p$ norm of perturbation and $\omega > 0$ is the weight associated to the amount of perturbation.

### 5.2.2 Accessibility Constraint

The objective function we discussed above considers that the attacker has the flexibility to modify any user/resource metadata to obtain their desired access. In MLBAC, there are metadata with different levels of restrictions, as discussed in Section 5.1 and illustrated in Figure 5.1. Therefore, an adversary can not access and modify every metadata equally. A sysadmin could explicitly impose such restrictions, and the adversary could not directly access and modify some of the metadata. For example, a user's 'job_role' information comes from the system, which is difficult, if not impossible, to modify and will take more effort. On the other hand, the adversary could influence some metadata, such as 'expertise', and maybe modify it with less effort. Without loss of generality, we refer to such notion of restrictions and the ability to access them as the *accessibility* constraint.

To impose the accessibility constraint while generating perturbation ($x_p$), we extend our objective function as below:

$$g(x_p) \;=\; \mathcal{L}(x + x_p,\, t) \;+\; \omega \parallel x_p \circ c \parallel$$

Where $c$ is the accessibility constraint expressing the magnitude of restrictions with respect to each metadata, $\circ$ is the element-wise product operator, and $\omega > 0$ is the weight for the penalty of perturbing metadata with higher accessibility constraint. We hypothesize that exploiting accessibility constraint with the proposed objective function will make the perturbation generation harder.

### 5.2.3 Adversarial Attack Approach

After formulating objective functions, we aim to develop an approach to optimize the objective function minimization problem and generate perturbed examples. Researchers have proposed numerous approaches, such as the Fast Gradient Sign Method (FGSM) [130], Projected Gradient Descent (PGD) [89], Carlini & Wagner Attacks [31], etc., to generate adversarial attacks that can deceive a trained deep neural network with higher accuracy. However, most of the algorithms have been proposed for the image domain. Due to the data characteristics in the access control domain, these algorithms may not readily mislead the ML model in MLBAC. A few works consider the non-image tabular data domain [14, 32, 88]. These works have demonstrated their potential in generating adversarial attacks for ML models applied in multiple use-cases such as financial services [14], fraud detection [32], etc.

Ballet et al. [14] propose the LowProFool algorithm to solve the optimization problem for the continuous tabular data using a gradient descent approach. As MLBAC input is tabular data containing both *categorical* and *continuous* metadata, and we have a similar optimization problem, we adopt their algorithm for minimizing our objective function $g$. We customize the algorithm and adjust the parameters such that we can optimize our objective function and supports continuous and categorical data. We further discuss this in Section 5.3.5.

## 5.3 Methodology

### 5.3.1 Determining Accessibility Constraint

In MLBAC system, the access decisions are made based on user/resource metadata. In practice, this metadata could come from different sources, where some sources are more secured than others. For example, a highly restricted database may store a user's sensitive information, such as 'job_role.' In contrast, a less secured database may store less sensitive data such as 'expertise.' Also, not every metadata/attributes equally influences the access decisions [109]. Therefore, the sysadmin may restrict influential and sensitive metadata/attributes with more security. We utilize the notion of accessibility constraint (as discussed in Section 5.2.2) to *simulate* such security restriction levels for each of the metadata.

Instead of randomly determining some metadata as more restricted and others as less restricted, we measure the correlation for each user and resource metadata for a decision. More concretely, we calculate the absolute value of Pearson's [18] correlation coefficient of each metadata with respect to the access decision. For this purpose, we could utilize other methods crafted specifically for the access control domain, such as the *global interpretation*-based method developed by Nobi et al. [109], where they determine the influence of each metadata for a specific decision for an ML model. However, we rely on Pearson's correlation as it can better simulate human intuition through a linear correlation based on the data [14]. For our proposed objective function, we generate accessibility constraint values in the range of 0 to 1 for each metadata. A higher constraint value of metadata indicates the respective metadata is more secured, thereby less accessible by the adversary while generating perturbation.

### 5.3.2 Dataset for MLBAC Adversarial Attack Experimentation

MLBAC makes access control decisions based on user and resource metadata and attributes values. Before that, MLBAC needs training with available *ground truth* to decide on subsequent access control requests. Generally, the ground truth could be access history, existing authorization information, etc., communicated through authorization tuples [30, 93, 109]. (We discussed the authorization tuple in Section 5.2.1.) We call such a collection of authorization tuples a *dataset* that represents the current access control state of a system.

Each dataset represents the metadata information of users and resources and their access control state. However, metadata values could be a *mix* of continuous (e.g., 'age,' 'salary') and categorical (e.g., 'security_level,' 'designation') data. We are unaware of any publicly available access control dataset representing mixed metadata values. There are two access control datasets from Amazon [10, 11]. Both of these datasets contain only categorical metadata/attributes values. As discussed in Section 3.3.1, we generate eight different datasets using the data generation algorithm proposed by Xu et al. [143]. Out of eight datasets, for this work, we experiment with $u5k$-$r5k$-$auth12k$ and $u5k$-$r5k$-$auth19k$ datasets.

We introduce different complexities to simulate some real-world scenario. Each dataset has around five thousand users, five thousand resources, and four different operations. The $u5k$-$r5k$-$auth12k$ dataset, we refer to as **System-1**, has around 12K authorization tuples. Each user has eight user metadata ($umeta0$, $umeta1$, $\ldots$, $umeta7$), and each resource has eight resource metadata ($rmeta0$, $rmeta1$, $\ldots$, $rmeta7$). On the other hand, the $u5k$-$r5k$-$auth19k$ dataset has around 19K authorization tuples, whereas a user has ten metadata ($umeta0$, $umeta1$, $\ldots$, $umeta9$) and a resource has ten metadata ($rmeta0$, $rmeta1$, $\ldots$, $rmeta9$). We refer to this dataset as **System-2**.

However, both datasets contain *categorical* metadata values, as integers, of which each value represents a category. In practice, one could anticipate a mix of continuous and categorical data [96]. To simulate the notion of mixed data, we consider metadata values of both datasets as mixed data. In particular, we assume that the first half of the user and resource metadata are continuous and consider an integer a real value. We also consider the rest of the metadata as nominal categorical data, where each integer represents a category, and the order of the category does not matter. As the metadata contains mixed data, they need to be processed efficiently such that the underlying ML model can properly consume the training data [57]. The following section discusses their preprocessing.

### 5.3.3 Data Preprocessing

This section explains the processing of the System-1, which is equally applicable to System-2. $\Big\langle$1011|2021|30 49 5 26 63 129 3 42 | 43 49 5 16 63 108 3 3 |$\langle$1 1 0 1$\rangle\Big\rangle$ is a sample authorization tuple of the System-1 dataset where 1011 and 2021 are the user and resource's unique numbers. The next eight elements indicate the metadata values of a user, the following eight elements represent the resource's metadata values, and the final four binary digits (1 for grant, 0 for deny) signify four different operation's access to the resource.

For our experiment, we consider the first four user metadata ($umeta0$ to $umeta3$) and the first

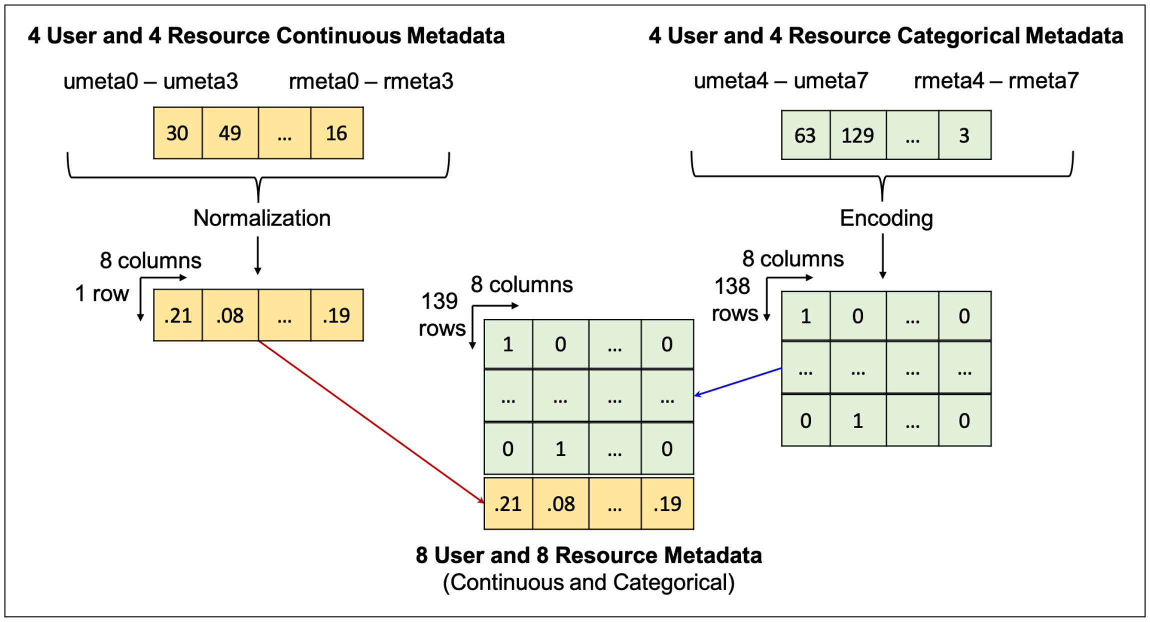

**Figure 5.2**: Input Preprocessing in System-1 Dataset.

four resource metadata ($rmeta0$ to $rmeta3$) to be continuous, and the rest of the metadata are categorical. Also, we utilize only *one* operation (out of four operations) that indicates whether the user has access to the respective resource or not (grant/deny). Figure 5.2 illustrates the data preprocessing of System-1 dataset. We normalize continuous data using the *MinMaxScaler* approach [145] and apply one-hot encoding [57] for the categorical portion of the metadata. As shown in the figure, we merge continuous normalized data, a vector, with the one-hot encoded categorical data matrix. The merged matrix, holding values between 0 and 1, acts as the ML model's input.

### 5.3.4 Candidate ML Model for MLBAC

A deep neural network has significant benefits for controlling access in dynamic, complex, and large-scale systems [109]. Considering the scale of our datasets and data characteristics, we utilize a deep neural network for MLBAC. In particular, we consider ResNet [59] as our candidate ML model. We note that one could use other deep neural networks, including Xception [35], DenseNet [69], etc., although we do not anticipate any significant changes in our results.

For System-1 and System-2 datasets, we exploit the ResNet ML models with 8 and 50 residual layers, respectively, as suggested in [109]. There is a convolution, a batch normalization operation [70], and a ReLU [103] activation function in each layer. The final activation layer's output is flattened and fed into a dense layer with a *sigmoid* activation function as the MLBAC needs to make a binary access decision (grant or deny).

### 5.3.5 Customization of LowProFool Algorithm

As discussed in Section 5.2.3, we exploit the LowProFool [14] algorithm for generating adversarial examples for the MLBAC. We modify the algorithm and parameters to support continuous and categorical data and help minimize our proposed objective functions discussed in Section 5.2.1 and 5.2.2. We replace the objective function with our proposed one and take accessibility constraint ($c$) as input. We define the loss function $\mathcal{L}$ as the binary cross-entropy function. Also, we guide the perturbation to the *positive* of the gradient if our target access is a 'grant (1)'. Otherwise, we guide to the *negative* of the gradient. As discussed, our experiment's preprocessed input holds values between 0 and 1. Therefore, we modify the $clip\,(x)$ function to control the value of each metadata such that it does not cross the range. We floor the value to 1 if it exceeds one, and we ceil it to zero if the value becomes negative.

In addition, the algorithm controls the growth of the perturbation using the scaling factor parameter $\alpha$. Also, we need to provide a weight parameter $\omega$ that helps to minimize perturbing the metadata with higher accessibility constraints. For the System-1, we choose the parameters $\alpha$ and $\omega$ as 0.2 and 6, respectively. These values are 0.7 and 5.7 for the System-2. We follow a trial-and-error process for choosing them. We have created a GitHub repository consisting of our experimentation's source code[1].

## 5.4 Evaluation

### 5.4.1 Evaluation Metrics

The efficiency of an adversarial attack is measured as the ratio of successfully crafted adversarial examples, and the total number of samples attempted to prepare the adversarial example. This

[1] `https://github.com/dlbac/MLBAC-AdversarialAttack`

ratio is known as the **Success Rate** [14]. Let us consider $\mathbb{X}$ as the number of samples attempted for the adversarial example creation. We also consider $\mathbb{A}$ containing every tuple ($x,\, x_a$) such that $x \in \mathbb{X}$ and $x_a$ a successfully crafted adversarial examples from $x$ with $f(x) \neq f(x_a)$. We define the Success Rate as follows:

$$Success\, Rate \;=\; \frac{|\mathbb{A}|}{|\mathbb{X}|}$$

### 5.4.2 Evaluation Methodology

We propose two different objective functions for the MLBAC system, as discussed in Section 5.2.1 and 5.2.2. In one case, we consider that adversary has equal access to every metadata and, thereby, could change any metadata at any level. In another case, we consider a scenario where each metadata may have been stored in places with different levels of restrictions (accessibility constraint). As a result, an adversary could not access and modify every metadata equally. In such a case, the adversarial example generation would be harder, and we anticipate a lower Success Rate. We evaluate both cases and refer to the first as the 'Without Accessibility Constraint' and the latter as the 'With Accessibility Constraint.'

As discussed, we experiment with two datasets System-1 and System-2, having a different number of samples and data characteristics. We evaluate the performance of both of the datasets. Also, we create a different subset of samples measured by sample count and eventually determine the average performance for the respective dataset.

### 5.4.3 Results

Figure 5.3 demonstrates the performance of System-1. As shown in the figure, the success rate of adversarial attacks without imposing any additional constraint is above 95%. However, when we restrict the system considering accessibility constraint, the success rate reduces to an average of 75%. We observe a similar result in the System-2 case, as demonstrated in Figure 5.4. In this case, without any constraint, the success rate is above 98%. We observe an identical trend for the scenario when we apply the constraint. The average success rate goes down to 79%. In both cases,

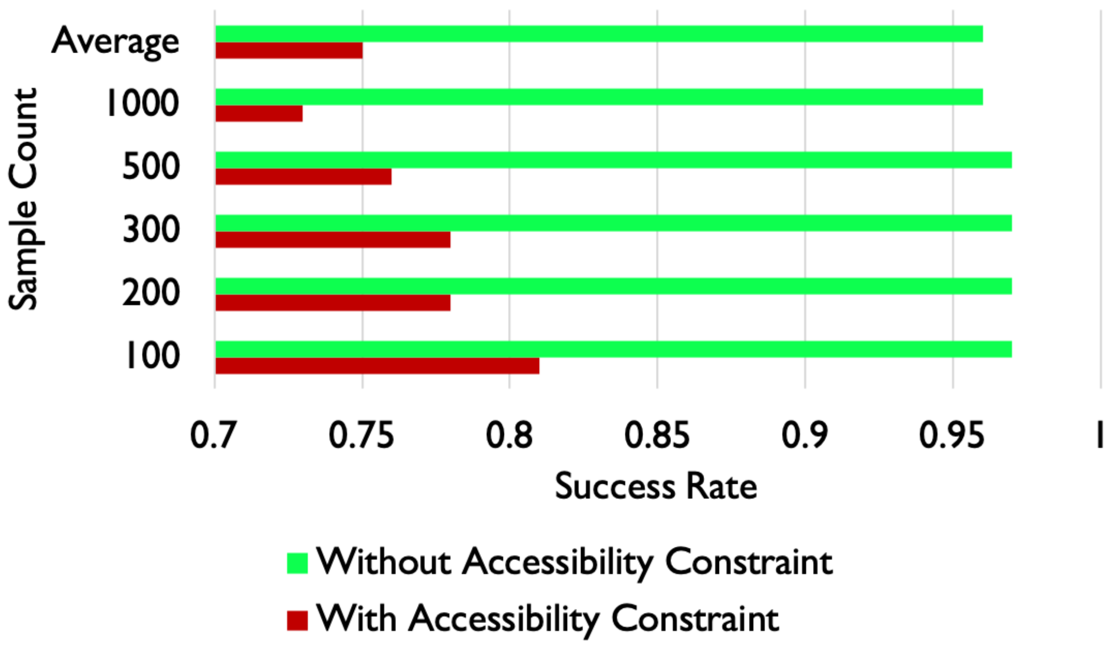

**Figure 5.3**: Performance Evaluation of System-1.

we observe that the proposed adversarial method could successfully design adversarial examples at a higher success rate. However, by restricting the flexibility of the adversary by limiting their access to different metadata at a different level, we see a dramatic reduction in the performance, justifying our hypothesis made in Section 5.2.2.

## 5.5 Conclusion

This chapter defines the adversarial attack problem in the MLBAC context where a trained ML model decides access. We customize objective functions by imposing additional accessibility constraint to reflect the access control context. We experiment with adversarial attack cases in the deep neural network-based MLBAC with two complex, large-scale systems. Also, we consider systems where underlying user/resource metadata contains both categorical and continuous data. We thoroughly evaluate the performance of MLBAC adversarial attack experimentation and demonstrate that it is possible to restrict an adversarial attack to some extent by introducing a security constraint.

In this work, we measure accessibility constraint using the correlation method. Exploring other methods to simulate such restrictions and accessibility would be interesting. Also, the adversarial

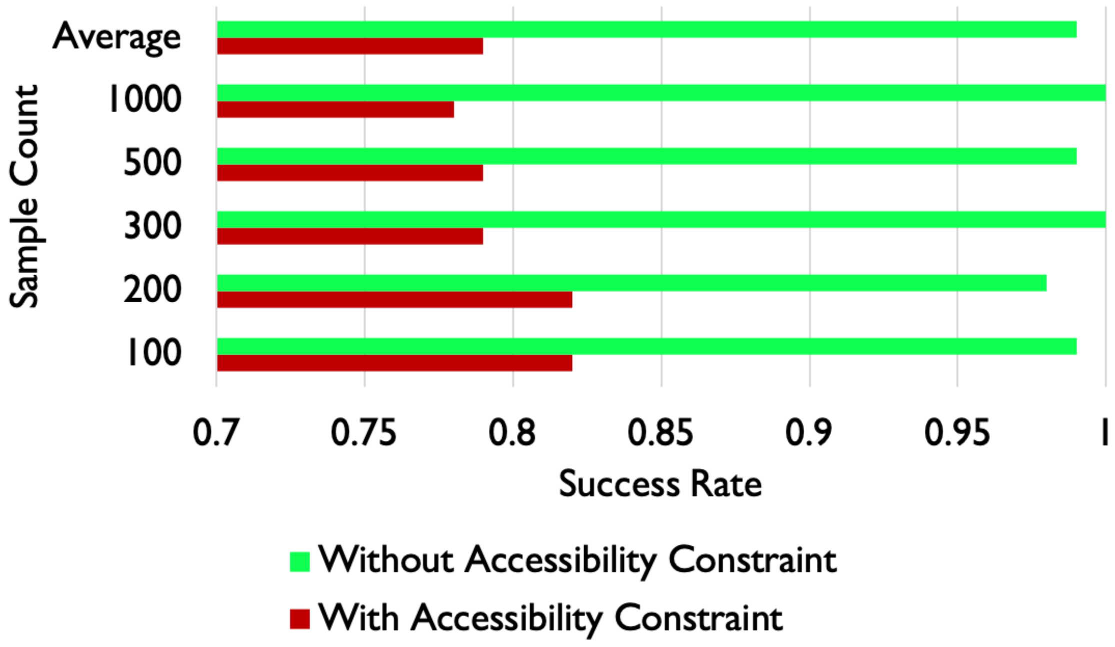

**Figure 5.4**: Performance Evaluation of System-2.

attack and the proposed objective functions depend on multiple parameters. The performance of the attack method has a strong connection with these parameter values, which we determine using a trial-and-error approach. A further investigation is needed to minimize dependencies and find their values more systematically. Besides, we consider the white-box attack scenario, where we assume the adversary has access to the ML model in MLBAC. It would be interesting to explore a black-box scenario where the adversary cannot access the MLBAC model. Also, the proposed accessibility constraint exhibits its prospect of safeguarding against adversarial attacks, which may not be sufficient in some cases. A thorough investigation is needed to determine defense techniques in the access control context.

# CHAPTER 6: IMPLEMENTATION AND EVALUATION OF DLBAC

Prior chapters of this dissertation thoroughly depict Machine Learning Based Access Control (MLBAC). As part of the operational model, Chapter 3 experiments with both classical ML-based approaches and DLBAC. As reported, the performance of DLBAC is significantly higher than the classical ML approaches and traditional access control systems. Further, Chapter 4 investigates an administrative model for MLBAC. The empirical results suggest that DLBAC has more advantages from an administration perspective. As a result, due to the performance benefits of a DLBAC for an access control system, we further implement DLBAC for recommending application permissions in mobile devices. Besides, we also explore how DLBAC could run in parallel with traditional access control systems.

## 6.1 DLBAC Assisted Permission Recommendation in Mobile Devices

### 6.1.1 Introduction and Motivation

Smartphones and wearable devices can perform very personalized tasks due to their high computing capability and access to numerous data collected from the user activities, various sensors of the device, online sources, etc. By combining this information, such as location, photos, messages, call log, contacts, calendar, etc., it is possible to extract high-level information about the potential activities of the respective user. While most of these data are very personal, the widespread adoption of smart devices reduces the protection-ability, and adversaries could collect this information for malicious use cases, which is a security and privacy concern [22].

Interestingly, without the help from the installed application (app) and proper access permission to information, collecting these data from smart devices is nearly impossible. Therefore, to restrict the access of devices' data, the first step is to avoid unauthorized apps, and the immediate next step would be to control the permissions. However, due to numerous sensors and services, the number of permission requests is very high, making it harder for users to choose and restrict them efficiently. The devices follow Ask-On-Install (AOI) or Ask-On-First-Use (AOFU) strategies to ease the permission management process at the app level. In the case of AOI, the user has to allow all the permissions to install the app or deny the installation. For AOFU, the app asks for permission for the first use. In either case, once approved, the app uses the earlier agreement for all future cases.

However, a permission decision is subjective and depends on the situation and actual needs (context). Due to the abundant permission requests from different apps and services, the users could not efficiently decide considering all these contexts, nor is it even practical to do for a human. As a result, human users either grant without knowing its further consequences or deny the request those maybe needed that may be required for a subsequent task. For automated permission decisions, researchers are actively studying this area to improve the efficiency of permission managers while considering users' personal preferences.

Generally, these automated processes work based on creating a set of profiles and assigning each user to a specific profile with respect to the user's preferences, device context, etc. For instance, authors in [3] and [111] develop methods to automate privacy decisions based on crowdsourcing. The proposed methods collect all the users' permission choices in a centralized domain and create one profile to recommend to users about their permission decision. Xie et al. [140] developed a location permission recommendation system for the Facebook [1] app using collaborative filtering. Ismail et al. [71] develop a privacy settings recommender system through various clusters in the context of Instagram [2]. The authors in [92] create a method with four different clusters that can predict a user's mobile app privacy settings. Also, for location-sharing apps, Ravichandran et al. [112] created profiles to predict the user's permission preferences that demonstrated better performance compared to the system without profiles.

Also, some approaches train an ML model based on prior preference, and the model could recommend granting or denying a permission request. Liu et al. [94] implemented a personalized permission manager (PPA) for recommending app permissions, and its recommendations were perceived as useful and usable. The authors evaluate the performance of the PPA with actual

[1] www.facebook.com

[2] www.instagram.com

mobile users using their own Android devices. The evaluation results suggest that users adopted approximately 79% of the recommendations made by the PPA, and they only changed 5% of the permission settings previously adopted based on the PPA's recommendations.

Recently, Brandão et al. [22] developed a privacy-aware permission preference system for mobile devices. The proposed system builds privacy profiles and predicts privacy decisions using the ML model while ensuring user privacy. The authors create privacy profiles exploiting the privacy-preserving clustering technique. In the proposed method, a server creates the profile without direct access to the user's data. Also, the authors apply the federated learning phenomenon for training the respecting ML model keeping user's data in user's device locally.

However, many of these research efforts generally focus on generating a single or a set of profiles based on the prior history of users' preferences (the dataset). While these approaches' performance looks promising, there is still a gap between actual preferences and the recommendation. As a result, this area needs more studies to develop a better recommendation system. This chapter investigates the potentiality of DLBAC for recommending permission decisions for mobile devices considering both users' and devices' contexts.

### 6.1.2 Dataset Characteristics

Mendes et al. [99] collected the permission preference data [3] from the Android users in the NGI Trust project COP-MODE4 through COP-MODE's Naive Permission Manager [4]. A sample in the dataset represents the user's permission decision (grant/deny) for a specific permission request from apps, stored as a JSON object. Each sample contains user id, permission dialog data, and context data. The user id represents the unique identification number of the human user and the id of the respective device. The permission dialog data represent the information of the requesting apps, such as whether the app is a foreground app or not, the app is a system app or not, the category of the apps, etc. Also, the permission dialog data contain the name of the requested permission, e.g., LOCATION permission, users' decision to the respective request, users' location, etc. On the

[3]https://cop-mode.dei.uc.pt/dataset
[4]https://cop- mode.dei.uc.pt/cm-npm

**Table 6.1**: List of Features.

| # | Name | Data Type | HasMissingValues |
|---|---|---|---|
| 1 | callState | Categorical | no |
| 2 | screenIsInteractive | boolean | no |
| 3 | networkStatus | Categorical | no |
| 4 | plugState | Categorical | no |
| 5 | selectedSemanticLoc | Categorical | no |
| 6 | category | Categorical | no |
| 7 | isTopAppRequestingApp | boolean | yes |
| 8 | isForeground | boolean | yes |
| 9 | isInEvent | boolean | yes |
| 10 | hour | Categorical | no |
| 11 | isWeekend | boolean | no |
| 12 | permission | Categorical | no |

other hand, the context data includes foreground and background running apps, network status, screen status, call state, etc. We refer to all these data as the *feature*.

The dataset comprises 93 users and 2180302 permissions requests, of which 65261 were manually decided (grant or deny) by the users. Of the 65261 permission requests, two-thirds of the requests were granted, whereas the rest of the requests were denied. The dataset contains requests from almost all the Android permissions. However, among all the permissions, there are two main permissions groups, LOCATION, and CONTACTS, representing approximately 50% of the requests. The PHONE and STORAGE permissions account for around 37% of the requests [22].

**Data Preprocessing**

For this experiment with DLBAC, we experiment with the manually decided requests (65261 permission requests). Also, context and permission dialog data contain a good number of features, some of which are not usable. For example, user id, device id, bootTime, answerType, etc., are static information that does not relate to permission decisions (e.g., bootTime, answerType) or is not helpful to generalize them across requests (e.g., user id, the device id). As a result, we consider a subset of features, e.g., callState, screenIsInteractive, networkStatus, permission name, permission decision (1 for grant, 0 for deny), etc. We finally transform a sample, which is stored as a JSON object, into the tuple of features and their values (permission is also part of the feature) and respective decision ($\langle$ *feature-value pairs, decision* $\rangle$). Note that we aim to train a model such that the trained model could predict the permission decision. In addition, each feature contains nominal categorical data. Note that some features contain boolean data (TRUE, FALSE). For simplicity, we consider them as categorical (two categories). We summarize all these features in Table 6.1.

**Privacy Profiles.** Prior research suggests that we can build some privacy profiles by exploiting the user and device's context. These profiles could help to identify privacy preferences at the user level [22, 94]. Also, such a profiling technique would help cluster like-minded users under the same group. We create *nine* such profiles across users using a hierarchical clustering algorithm and apply those profiles as part of the features. We will show that extracting this profile information and adding to the features help obtain better performance.

**Conflict Resolution.** Apparently, there are scenarios where there might have a entirely opposite permission decision for the same feature-value pairs. For example, user Alice once *grant* for CONTACTS permission requested by an apps, say $X_{apps}$, and *deny* for CONTACTS permission for another apps, say $Y_{apps}$, having *same* features (e.g., category, isSystemApp, etc.). Even though, the requests are from two independent apps, as the apps have same features and same user Alice is once granting and later denying the same permission, the decision is contradictory. We term this as a *conflict*. In such a case, we resolved this using the *permit-overrides* strategy [91] that is– if the Alice grants for a permission at any episode, we consider Alices' decision as the grant for that permission even if there are cases where Alice denies for the request comes from different apps with same features.

**Missing Data.** Some features contain missing values, e.g., isTopAppRequestingApp. In practice, this is very likely that all the apps may not have the same number of features. Also, there might be limitations in collecting some features, e.g., the calendar may not be synced to collect event data. Therefore, it is required to consider such requests while training and evaluating the model's performance. Standard techniques are available in the literature to fill these missing values, such as imputation strategy [5], considering missing values as missing or unknown, etc. For this piece of work, we replace missing values with a new category as 'unknown' that better reflects the actual scenario.

### ML Model Training

For a given feature vector $x$ of the feature-values pair, the DLBAC access control decision making can be defined as a prediction function $f$ such that $\hat{y} = f(x)$, where $\hat{y}$ is the predicted decisions (grant (1) or deny (0)) for the requested permission.

Since the feature values in our dataset are categorical, we need to map them to binary values before providing them as input to DLBAC, as discussed in Section 3.3.3. We encode feature-value pairs using *one-hot encoding* [57] to transform the categorical values into a two-dimensional binary array. The row in the array represents features, and the column holds the encoded binary representation of the corresponding value.

### Evaluation Methodology

As evaluated in Section 3.4.1 for the DLBAC, for this evaluation, we also created three instances of DLBAC for this experiment using three deep neural networks, including ResNet [59], DenseNet [69], and Xception [35]. Also, we use the same architectures, parameters, and configurations suggested in the same section.

Besides, we follow the same evaluation metrics (F1 Score, True Positive Rate (TPR), False Positive Rate (FPR), Precision, and accuracy) discussed in Section 3.4.1. For details of the metrics, we refer to the respective section.

Using the same dataset, Brandão et al. [22] developed a privacy-aware permission preference

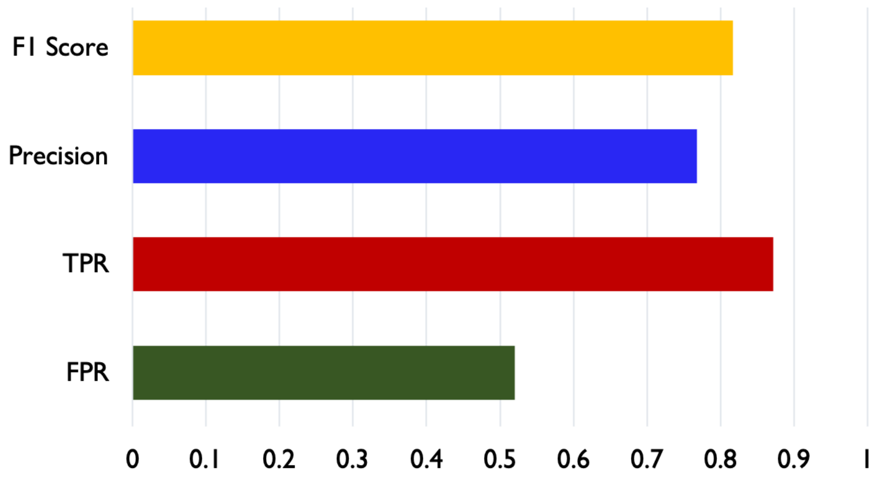

**Figure 6.1**: Performance of DLBAC ResNet Instance Without Privacy Profiles.

system for mobile devices. As the current state-of-the-art, they reported the model's accuracy as 0.88 and F1-score as 0.90. We compare our experiment's performance with theirs.

### Results

First, we evaluate the performance of the DLBAC ResNet instance. In this experiment, we did not consider privacy profiles. Our experiment shows that, on average, DLBAC obtains an accuracy of 74.02% and the average F1 Score of 0.82. We also determine the FPR, TPR, and Precision, as shown in Figure 6.1. As we see in the figure, the FPR is quite high (> 0.5), indicating the model is over-privileged [109]. Though the TPR is reasonable, the Precision is quite low. Such a result indicates the model is not efficient in granting the permissions [109].

Next, we consider the privacy profile information as part of the features and create three DLBAC instances using the same features. Figure 6.2 demonstrates the accuracy of different instances. The accuracy is consistent across instances with an average of 88.5%, a significant improvement from the model without the privacy profiles. Also, we determined the F1 Score, FPR, TPR, and Precision for all the instances and reported in Figure 6.3. The F1 Score sees a substantial improvement. Also, the TPR and Precision increased significantly while the FPR decreased to, on average, 0.177. Such an improvement with higher TPR and lower FPR indicates that these instances can efficiently balance both over- and under-privilege [109].

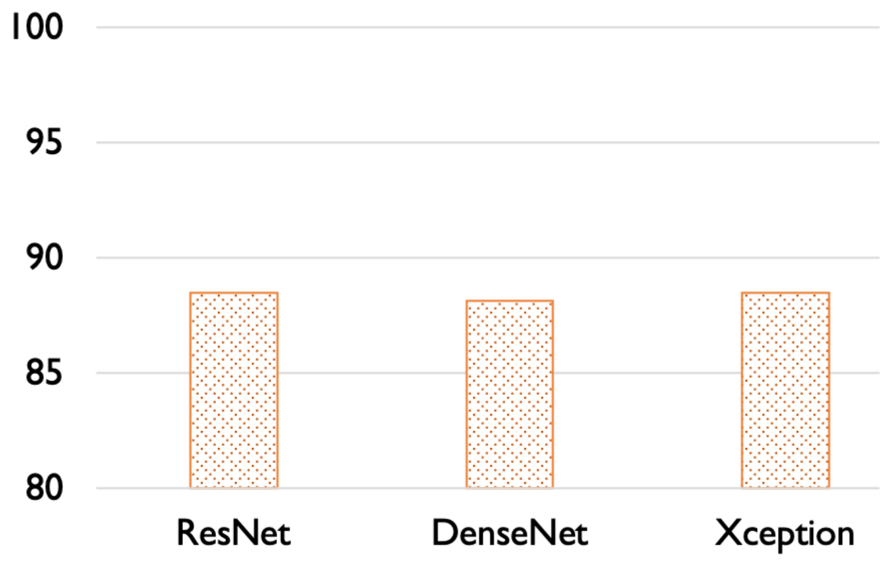

**Figure 6.2**: Accuracy Comparison Among DLBAC Instances.

Overall, the performance results demonstrate the efficiency of DLBAC in recommending (or even automatically deciding) mobile devices' permissions.

## 6.2 Other Potential Applications of DLBAC

### 6.2.1 DLBAC in Tandem with Traditional Access Control System

In practice, we do not advocate that DLBAC will immediately substitute traditional access control forms. On one side, DLBAC shows promising results compare to traditional access control systems. On the other side, the operation of DLBAC is entirely different than existing access control practices. System administrators are used to functioning access control states in terms of written policies, which is quite acceptable considering the sensitivity of the domain. One obvious issue is that DLBAC is a black-box neural network-based system. The administrator can not observe the underlying policy to know how DLBAC controls its access control states. They might raise different concerns about the usability, practicality, or trustworthiness of DLBAC, which is entirely rational. Even though DLBAC provides extensive tools and techniques to configure and manage access control states of a system and understand access control decisions, it might not be sufficient for many applications. Therefore, it is not unusual to be skeptical about the applicability or practi-

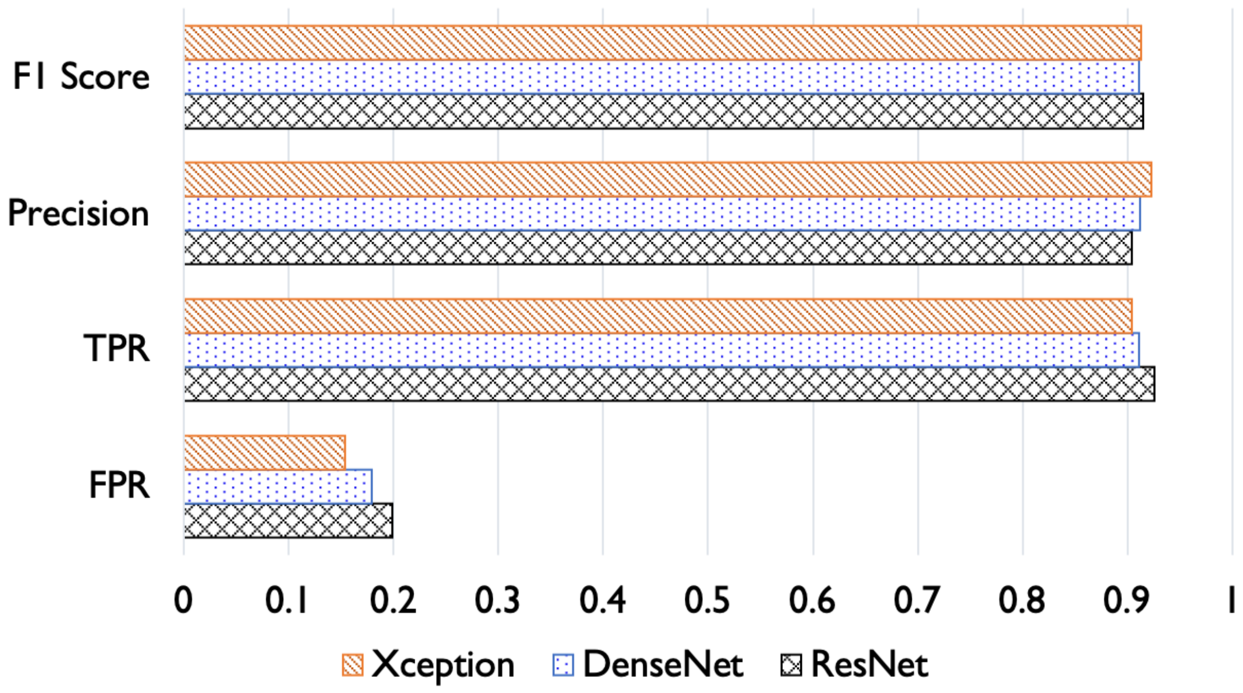

**Figure 6.3**: F1 Score, Precision, TPR, and FPR Comparison Among DLBAC Instances.

cality of the DLBAC. A direct resolution could be to apply DLBAC in parallel with the traditional access control system, which could be a research challenge.

This section discusses two unique approaches to how DLBAC could be effectively integrated to operate in tandem with traditional access controls such as RBAC or ABAC. Note that they are not mutually exclusive and can be deployed simultaneously.

**Reinforcing Access Control Decision**

We have already demonstrated that DLBAC can make better access control decisions than the traditional access control model through experimentation and evaluation. In addition, traditional policy-based systems are not as generalized as DLBAC, which is also proven through our experiments. Things worsen with– (1) the change in access control policy and user/ resource attributes and (2) when it is required to make access control decisions for unknown users or resources that were not explicitly seen during mining. The former is an administration issue in a traditional access control system, whereas the latter is a generalization problem.

We discussed in previous chapters that the traditional access control systems' administration

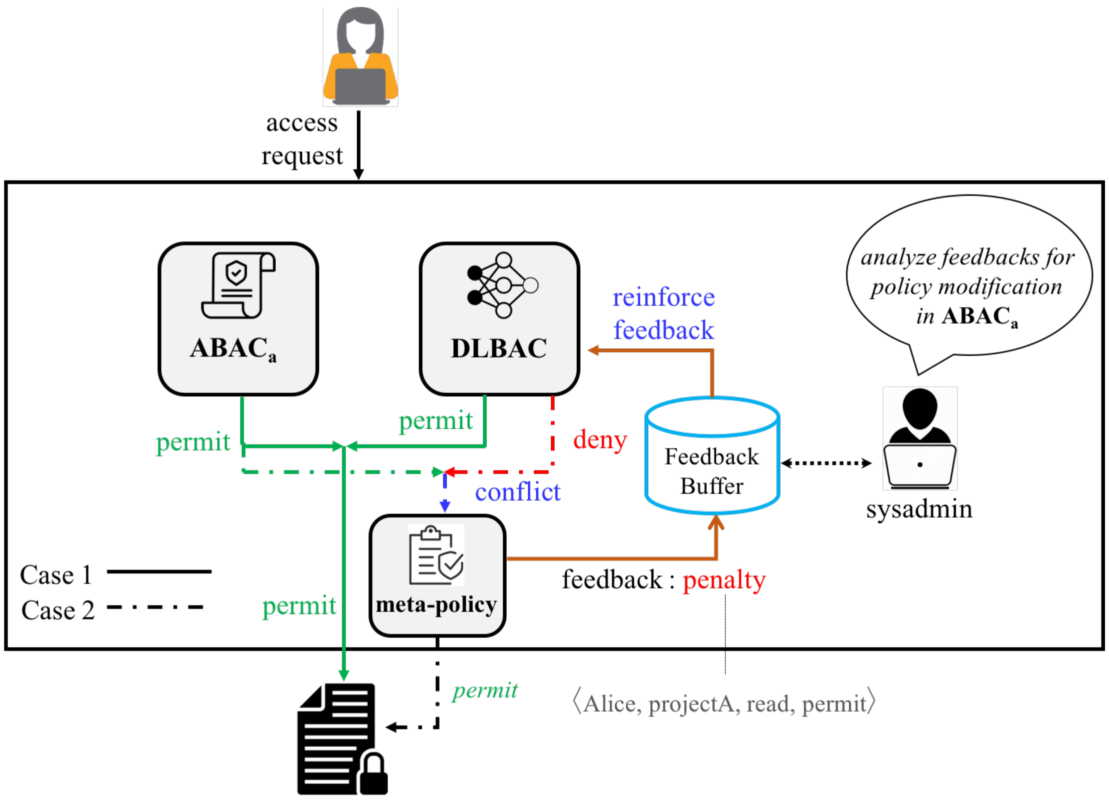

**Figure 6.4**: Reinforcing Access Control Decision.

is quite error-prone as a human manages them. Besides changing the access control policy, the administration may modify multiple dependent elements such as attributes assignment, role assignment, etc. Such a process is burdensome and quite challenging due to the lack of proper tools or techniques to control the process [138]. Eventually, human administrators ended up with a system that approves unauthorized access or denies legit requests. That makes overall policy maintenance inefficient and ineffective and introduces many errors in the policy. Therefore, after each policy maintenance, the system may expect a lot of unauthorized or unreasonably restricted access.

Parallel deployment of DLBAC, being a more generalized access control model, can help to improve performance in a traditional system, say $\mathrm{ABAC_a}$. Each access control request will go through DLBAC and the $\mathrm{ABAC_a}$ for a decision. This can be seen as a *two factors authorization* of access control decision. While the $\mathrm{ABAC_a}$ may make initial access control decisions, the DLBAC can be plugged as the second factor. A request would be granted or denied without any question if both factors agreed to do so. However, if their decisions do not match, a *conflict* arises, indicating a mismatch between DLBAC and the $\mathrm{ABAC_a}$ method. Ideally, an expert's (e.g., a system admin) intervention is required to resolve such conflict. Resolving every conflict with a human administrator might not be practical as, even in a medium-sized system, there might be many conflicts between DLBAC and other traditional models.

To minimize such human intervention to a large degree, we proposed establishing a *meta-policy* to automate such conflict resolution on how these two independent approaches will work together. The policy will act as an expert here and determine in what circumstances what should get preference or when to notify the system administrator to resolve a conflict, if not possible, according to the current meta-policy. For example, a straightforward meta-policy could be to *accept* any decision made by the $\mathrm{ABAC_a}$. Another example could be to take the DLBAC decision only if it is decided with more than 99% confidence (probability of DLBAC output is 0.99).

As shown in Figure 6.4, DLBAC is deployed in parallel with a traditional access control system, $\mathrm{ABAC_a}$, to reinforce access decisions. The figure illustrates two different scenarios. In the first case, both DLBAC and the $\mathrm{ABAC_a}$ made the 'permit' decision for a request, and the user is permitted to access the desired resource. The $\mathrm{ABAC_a}$ made a 'permit' decision in the second case, whereas the DLBAC denied the request, creating a conflict. Then, to resolve the conflict, the request goes through the meta-policy. The meta-policy takes the access request, decisions made by both $\mathrm{ABAC_a}$ and DLBAC, and then determines that the request should be permitted.

Evidently, a conflict in the system indicates that the decision in both DLBAC and $\mathrm{ABAC_a}$ is not consistent with the actual access control states. Therefore, adjusting the DLBAC and $\mathrm{ABAC_a}$ is equally important to minimize these conflicts. The system communicates a conflict with the traditional approach and DLBAC by providing a *feedback*. This feedback phenomenon can be considered a *human-in-the-loop* for strengthening underlying approaches capability.

**Feedback For DLBAC.** Feedback is an authorization tuple made from the access request and the access determined by the meta-policy in the case of conflict. As a result, feedback can be considered as *reward* from the DLBAC perspective if the authorization tuple is made from the access request and DLBAC's decision. On the contrary, feedback can be regarded as a *penalty* if the authorization tuple is made from the access request and the decision made by the $\mathrm{ABAC_a}$.

Suppose the meta-policy asserts that DLBAC made the correct decision for a request. In that case, the administrator will reward DLBAC and add feedback to a *Feedback Buffer*. If the conflict were to be resolved against DLBAC, the administrator would *penalize* DLBAC and add the tuple to the buffer. The feedback stored in the feedback buffer is incorporated into the DLBAC network during the DLBAC administration to *reinforce* the learning such that the DLBAC can decide with more confidence for any future request. Figure 6.4 illustrates that meta-policy accepted the 'permit' decision made by the $\mathrm{ABAC_a}$, and it provides feedback to the Feedback Buffer. In this case, the feedback is a *penalty* as the conflict resolution went against the DLBAC's decision. Eventually, all the stored feedback from the buffer is incorporated into DLBAC.

**Feedback For Traditional System.** The system administrator could also analyze the feedback from the buffer to determine any potential changes in the traditional system. For example, for some requests, DLBAC always denies access while the $\mathrm{ABAC_a}$ permits it. This might happen due to any access leaks (unauthorized access) in $\mathrm{ABAC_a}$ policy that probably occurred due to any recent changes in $\mathrm{ABAC_a}$. Then, the system administrator might consider adjusting the policy of $\mathrm{ABAC_a}$ to avoid related unauthorized access.

Overall, such a dual-authorization system will help build a reliable system that can make access decisions with more confidence. As there is a feedback mechanism for the conflicts, both DLBAC and traditional methods can improve their capacity, eventually helping to make better access control decisions. Also, the System administrator's active involvement in the process could back up to minimize the system's error and improve efficiency over time.

### Facilitating Access Control Administration

As discussed in the previous section, traditional access control administration suffers from numerous inefficiencies due to the lack of proper tools and techniques. This section proposes DLBAC as a candidate administration tool for the traditional access control system that can effectively guide

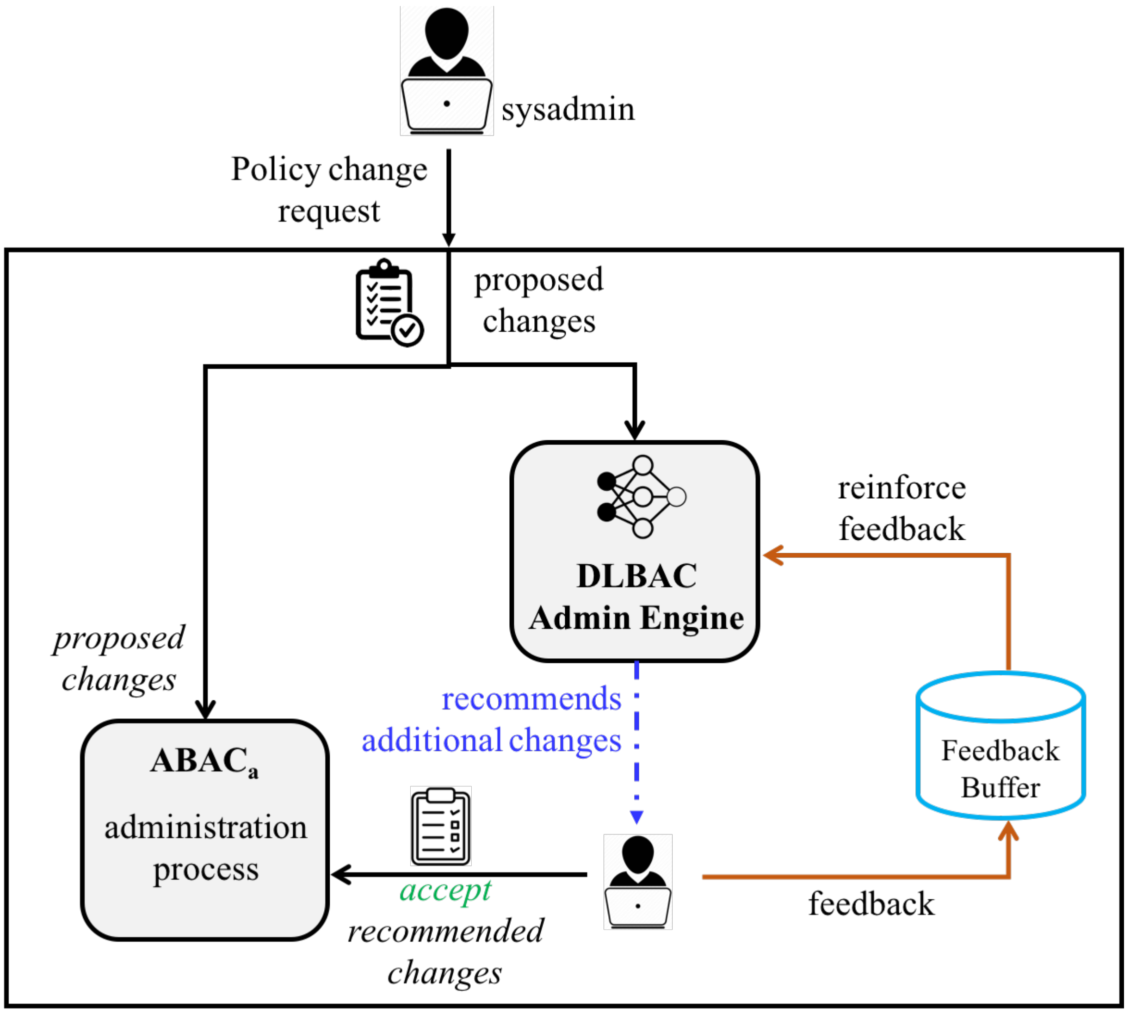

**Figure 6.5**: Facilitating Traditional Access Control Administration.

and ease the administration process.

For example, the user *Alice* doesn't have access to the *sub-project-list-api* resource. Due to the business policy, the system administrator wants to grant *read* access for *Alice* to this resource. Suppose no additional tools have been applied in a traditional access control system. In that case, it is hard to determine other affected users/resources due to the mandated change, and they are impacted by either losing existing accesses or gaining unauthorized accesses. In a traditional access control system, part of this problem is considered the side effect of access policy changes and is widely known as *access-denied* issue where deserving access is denied falsely [142]. A human administrator is not alone sufficient to tackle such an issue as it might create additional errors. DLBAC can help in this circumstance if deployed in parallel. The admin engine of the

DLBAC can identify and suggest other impacted users or resources due to a projected change in the system.

In addition, the access of *sub-project-list-api* resource may have dependencies to other resources, say, *main-project-list-api* which is, say, a gateway API before accessing underlying APIs. Without having *read* access to this resource, the newly added access will have no impact for *Alice*. In such a case, DLBAC can recommend what other accesses should be considered before committing the new change. Indeed, the system administrator may entirely or partially *accept* the recommendation or completely *deny* what DLBAC recommended. Besides, to make DLBAC more efficient and make more accurate recommendations in the future, the system administrator will provide feedback to the DLBAC. The idea of the feedback here is similar to what we discussed in Section 6.2.1. The only difference is instead of a single authorization tuple, in this case, feedback could be multiple tuples depending on what DLBAC recommends. As illustrated in Figure 6.5, DLBAC Admin Engine suggests additional changes beyond the proposed changes, and the system administrator accepts the recommendation. As a result, the administration process of the $\mathrm{ABAC_a}$ system received both proposed and recommended changes, eventually incorporating them while adjusting the policy. Also, the feedback is reinforced in DLBAC to improve its confidence for future recommendations.

Overall, such tooling support will make the administration process in a traditional access control system more intelligent, generalized, and efficient. Also, with experts' feedback, DLBAC can make more efficient recommendations.

### 6.2.2 DLBAC in Internet of Things (IoT)

With the proliferation of smart devices, the need for IoT ecosystems like Samsung SmartThings, Google smart home, Amazon echo plus voice assistant, etc., are also in high demand. One of the main goals of these human-centered IoT devices is to improve human awareness of their surrounding environment, improve user experience, and save cost in terms of time and money [126]. The primary communication happens between machines vs. humans through client-server interactions.

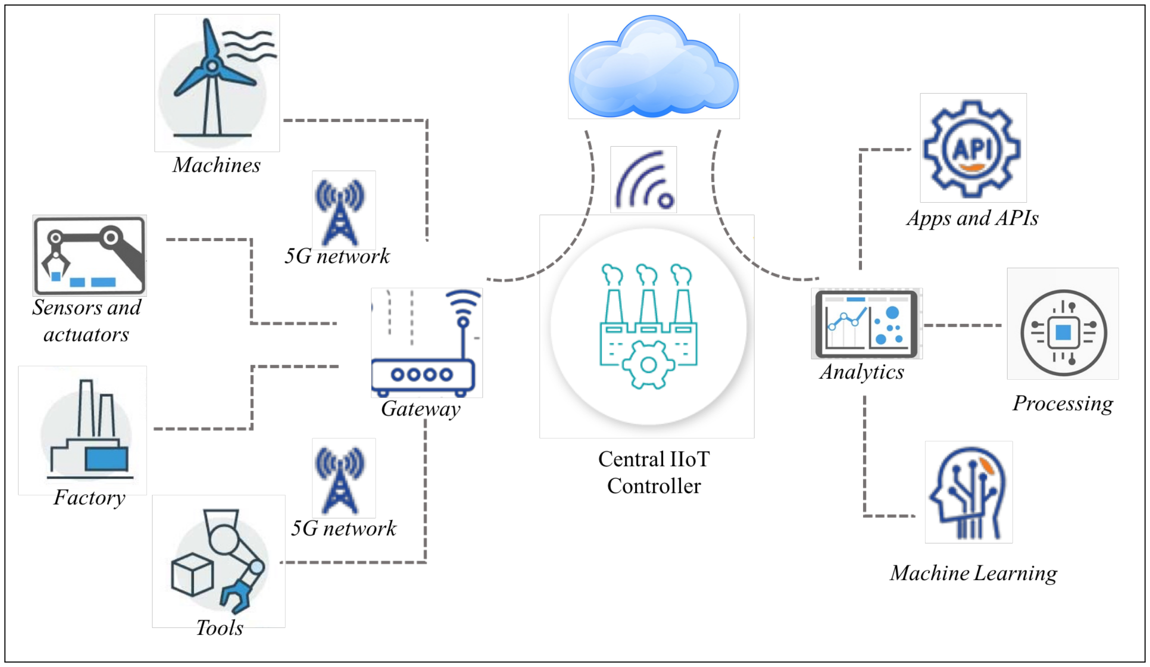

**Figure 6.6**: Typical Industrial Internet of Things (IIoT) Pipeline.

Thus, managing such a small setup is trivial in many aspects.

In the current world, the IoT goes beyond the typical consumer-level use cases and is extensively adopted in industrial sectors and applications in the name of industrial IoT (IIoT). In short, the Industrial Internet of Things (IIoT) can be defined as the use of IoT technologies in manufacturing [2]. In IIoT, as shown in Figure 6.6, thousands of sensors, instruments, and devices are connected with multiple cloud services and dependent on numerous communication protocols, WiFi, Bluetooth, LoRa (Long-Range), etc [21]. The consistent capturing and transmitting data among smart devices and machines generate large and varied data giving industries considerable opportunities to make decisions by analyzing all those information. Constant communication among these smart *things* allows for data and information exchange and analysis to improve productivity and operational efficiency. The manufacturer can combine machine data from a single device to a group of devices such as assembly facilities and manufacturing plants, which helps to identify the faults in the devices, lacking in the production processes and predict quality issues. Also, by deploying IIoT, the industries predict potential maintenance of the instruments to enhance the overall effectiveness and reduce maintenance costs.

The IoT includes connecting heterogeneous networks through various communication technologies and devices, complicating the ecosystem. While the most general communication requirements of IoT and IIoT are similar, issues like scalability, availability, reliability, and security and privacy are specific challenges for IIoT [126]. Also, to ensure more efficient production and unique services, the current design of IIoT emphasizes the possible unification and interconnection of manufacturing plants and machinery that may not be physically connected. With the invention and faster adaptation of the fifth-generation cellular (5G) network, it is envisioned that the 5G technology could be a potential solution to ensure the *availability* of IIoT services apart from other technologies.

However, the acceptance and success of all the efforts in IIoT technology are highly dependent on how well it can manage the information security and data privacy protection of all of its components [38]. One of the critical security aspects of IIoT is to ensure efficient identification and authorization mechanisms in the IIoT ecosystem so that only authorized users or entities can access the IIoT resource. Due to the dynamic entities and complicated communication, existing access control and authorization solution may not be applicable or efficient for the modern IIoT. For example, it is pretty difficult for a traditional access control system to analyze such a complicated system and design a comprehensive access control policy to manage the access authorization of all these heterogeneous entities. Over time, these IIoT devices will become more smart and intelligent to work with more significant memory, processing, and rationalizing inclinations in the future [55]. To tackle such a dynamic, creating artificial intelligence-enabled Internet of Intelligent Things could be one of the potential solutions [13]. DLBAC, being a deep neural network-based solution, has the power to observe, learn, and react against any access request, even if it is not explicitly seen during the training of DLBAC. As demonstrated in Section 3.4.2 of Chapter 3, DLBAC's performance is particularly pronounced in complex systems with a large number of authorization tuples. Such outstanding results of DLBAC justify the potentiality of DLBAC in an IIoT setup. DLBAC could learn underlying access relationships and access dependencies among entities in IIoT and provide access authorization decisions more intelligently.

## 6.3 Conclusion

This chapter discusses the application and implementation of DLBAC to solve real-world problems. First, we exploit DLBAC to recommend mobile devices' permission requests automatically. The performance evaluation suggests that DLBAC can effectively recommend required permissions while considering users' privacy preferences. Next, we discuss other potential use cases of DLBAC in a complex real-world system where a classical access control system may not be feasible (e.g., IIoT). Also, we demonstrate the potentiality of DLBAC that could help run in tandem with classical systems to manage them efficiently.

# CHAPTER 7: FUTURE DIRECTIONS AND CONCLUSION

## 7.1 Future Research Directions

In this section, we discuss some of the challenges for MLBAC and explore some ideas of how those could be addressed.

- **Verification.** A reliable system design is usually ensured through a rigorous *testing* and *verification*. Testing evaluates a system in several conditions to observe underlying behavior and detect errors. In contrast, the verification ensures that the system will not demonstrate any misbehavior under more general circumstances [52]. Likewise, in access control, the implemented policy is verified and tested [65, 67, 86] akin to verifying the correctness of software functionality (i.e., testing). It is required to ensure that the engineered access control rules make correct access control decisions, which is complex and needs expensive effort. Unlike many domains, failing to verify the access control system's correctness to a large extent may have serious consequences, including but not limited to over-provision, under-provision, adversarial attacks, etc.

  The MLBAC system's performance is measured based on an unseen *test dataset* to test the learned policies' correctness to some degree. Evidently, this testing method cannot find all possible cases that may be misclassified. As a result, a comprehensive verification of MLBAC is essential to deploy in security-critical systems. Interestingly, MLBAC testing and verification can be defined as a machine learning model's verification and testing problem. There are methods to verify a machine learning model automatically [90]. Also, each method has merits and demerits, and no one method will work for all the domains. Consequently, further research is needed to design a systematic verification and testing framework for MLBAC from machine learning and access control perspectives.

- **Bias and Fairness.** As MLBAC systems learn based on metadata distributed across various parts of an enterprise, there might be different types of human biases or errors in training data. As such, the MLBAC network trained based on such data can inadvertently be biased, favoring some decisions. For example, as we observed in the $amazon\text{-}kaggle$ dataset, most of the authorization tuples were with 'grant' decision, and DLBAC$_\alpha$ instances were biased towards the same decision. Moreover, data preprocessing could unknowingly introduce some biases. For instance, as discussed in Section 6.1.2, we rely on the *permit-overrides* strategy while resolving a conflict. One could follow the opposite method by adopting the *deny-overrides* strategy. Both are standard approaches [91]; however, the former could bias the model toward permitting access while the latter could bias toward denying access. Therefore, to obtain a fair and trustworthy MLBAC system, it is critical to audit training data, evaluate decisions for fairness, and establish a proper feedback loop [98].

- **Adversarial Attack.** Section 3.5.1 in Chapter 3 demonstrates that one could exploit Integrated Gradient-based understanding to change users' accesses by modifying metadata values that are influential for a specific decision. On the one hand, this is an advantage from an administrator's perspective to understand a decision and use its power to change others' access. On the other hand, it could be a security issue if a third party or an adversary has the information of influential metadata and access to respective metadata. Also, Chapter 5 discusses adversarial attack issues in MLBAC. The experimentation shows that it is possible to gain unauthorized access by slightly modifying the metadata values. However, experiments demonstrate that ensuring access restrictions for the metadata could significantly reduce the adversarial attack.

  Therefore, the first step toward minimizing or stopping the adversarial attack is to secure the metadata. It needs to study further vulnerabilities in MLBAC. Also, in our experiments, we only considered the DLBAC system. Exploring the potentiality of adversarial attacks is required if the ML model in question is a classical ML model such as Random Forest.

- **Understanding Access Control Decisions.** In Chapter 3, we investigate classical ML approaches and DLBAC. The accuracy and generalizability of DLBAC are significantly higher

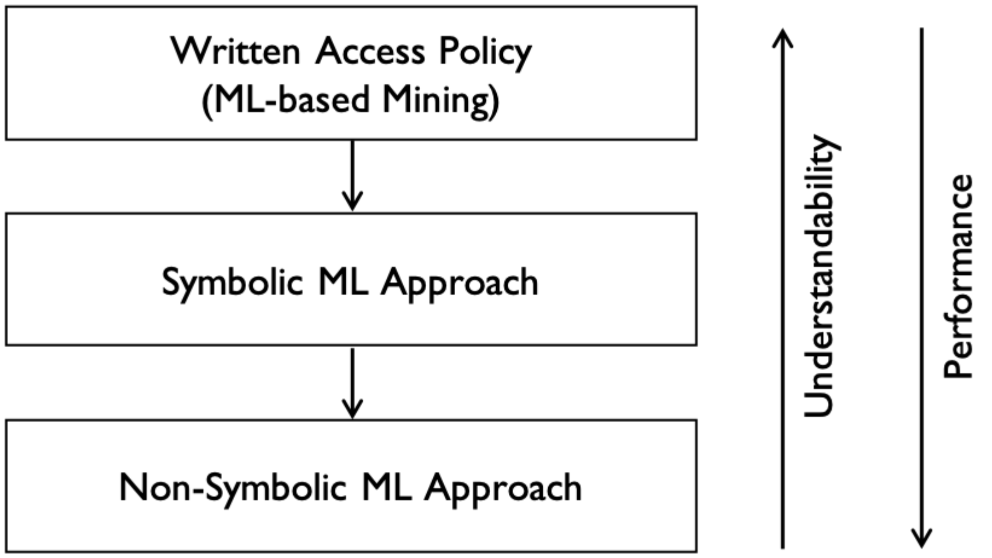

**Figure 7.1**: Trade-off Between Performance and Understandability in ML-based Systems.

than other ML-based approaches such as Random Forest (RF) and traditional access control systems such as ABAC. Along with accuracy performance, it is equally important to understand why an access control system decides for a specific request. Traditional access control systems, e.g., ABAC, have very well-written rules justifying the rationale of an access decision. In the case of a symbolic ML-based system (e.g., RF), one can dig down its underlying tree to extract logical rules and learn why a particular model makes a decision [30]. In contrast, in non-symbolic ML model-based systems, e.g., DLBAC [109] or Karimi et al. [77], a 'black box' function makes access control decisions and is not in a human-understandable form [30, 124]. Even though the use of ML models for access control decisions is steadily emerging, a lack of understandability could hinder the growth of this potential direction.

As illustrated in Figure 7.1, as we go from the written policy to the non-symbolic approach, the performance goes up, though with the cost of a lack of understanding of decisions. This is one of the major limitations when choosing a non-symbolic ML model for access control. This issue applies to other domains, including computer vision, malware analysis, financial systems, etc. The understanding issue, also known as interpretation or explainability in the typical ML arena, is a very active research area [**?**]. In many cases, it is observed that the solutions are domain-specific. For example, a *computer vision* specific interpretation method would not directly work in the access control domain.

In Section 3.5, we explore this issue for the first time from access control perspectives and provide two methods for understanding neural network-based access control decisions in human terms to a large degree [109]. However, the proposed methods do not guarantee the understanding of decisions with 100% accuracy. Also, the proposed methods may not be well suited across applications/ systems and may require more accurate and useful understanding. Therefore, a gap between understanding and performance needs to be minimized and further explored.

- **DLBAC in Tandem.** Section 6.2.1 thoroughly discusses the different use cases of DLBAC working in parallel with a traditional access control system such as ABAC. As discussed, while reinforcing access control decisions in ABAC, there might have conflicts. In such a case, we propose to develop a meta-policy that could help overcome such a conflict. In this case, the overall success of such settings will rely on designing a better meta-policy. Also, developing an efficient feedback system is critical in this scenario. Therefore, it needs to investigate further in a real-world setting to determine the feasibility of operating DLBAC in tandem with traditional access control systems.

In addition, the current proposal entangles a human in the loop in the case of classical systems' administration use. For larger systems, e.g., IIoT, such human involvement may not be practical. As a result, it needs to examine such applications further to determine a feasible system.

## 7.2 Conclusion

In this dissertation, we propose DLBAC, a deep learning based access control approach, as part of machine learning based access control. DLBAC shows the potential to deal with issues in classical access control approaches. As DLBAC learns based on metadata, it obviates the need for attribute/role engineering and updates, policy engineering, etc. We implement DLBAC$_\alpha$, a prototype of DLBAC, using both real-world and synthetic datasets with varying complexities. We demonstrate DLBAC$_\alpha$'s effectiveness and generalizability across different datasets. We propose two unique techniques to understand DLBAC decisions in human terms. Moreover, we investigate MLBAC administration issues and propose novel methods to perform access control changes in the system. We validate the effectiveness and quality of MLBAC administration by implementing two prototypes. There are some challenges related to the administration, and we propose various strategies to overcome them. Overall, the administration experimentation results suggest that a DLBAC could capture access control changes effectively while retaining the existing access control state.

Besides, we investigate the vulnerabilities in MLBAC in an adversarial setting and design a method to defend against malicious attacks in the context of access control. We demonstrate that it is possible to reduce adversarial attacks to some extent by employing access control related constraints. Moreover, we demonstrate the efficiency of DLBAC in complicated real-world settings. In particular, we implement DLBAC to recommend the permission decisions of different apps on mobile devices. We experiment with a real-world dataset collected from real Android phone users. Our evaluation results suggest that a DLBAC could recommend granting or denying permission with 88.5% accuracy. In addition, we also demonstrate the potentiality of DLBAC to operate in tandem with traditional access control systems and discuss how effective a DLBAC system could be in critical scenarios like Industrial IoT (IIoT). Finally, we discuss how to address current challenges in MLBAC and DLBAC and highlight future research directions.

# BIBLIOGRAPHY

[1] IBM (2004). Course registration requirements., 2004.

[2] L Aberle. A comprehensive guide to enterprise iot project success. iot agenda, 2015.

[3] Yuvraj Agarwal and Malcolm Hall. Protectmyprivacy: detecting and mitigating privacy leaks on ios devices using crowdsourcing. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 97–110, 2013.

[4] Ashraf Alkhresheh, Khalid Elgazzar, and Hossam S Hassanein. Adaptive access control policies for iot deployments. In *IEEE IWCMC*, 2020.

[5] Paul D Allison. Multiple imputation for missing data: A cautionary tale. *Sociological methods & research*, 28(3):301–309, 2000.

[6] Manar Alohaly et al. Towards an automated extraction of abac constraints from natural language policies. In *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer, 2019.

[7] Manar Alohaly, Hassan Takabi, and Eduardo Blanco. A deep learning approach for extracting attributes of abac policies. In *ACM SACMAT*, 2018.

[8] Manar Alohaly, Hassan Takabi, and Eduardo Blanco. Automated extraction of attributes from natural language attribute-based access control (abac) policies. *Cybersecurity*, 2(1):1–25, 2019.

[9] C. AL Amaral, Marcelo Fantinato, Hajo A Reijers, and Sarajane M Peres. Enhancing completion time prediction through attribute selection. In *Information Technology for Management: Emerging Research and Applications*. Springer, 2018.

[10] Kaggle Amazon. Amazon employee access challenge in kaggle, 2013.

[11] UCI Amazon. Amazon access samples data set, 2011.

[12] Luciano Argento, Andrea Margheri, Federica Paci, Vladimiro Sassone, and Nicola Zannone. Towards adaptive access control. In *DBSec*. Springer, 2018.

[13] Artur Arsénio, Hugo Serra, Rui Francisco, Fernando Nabais, João Andrade, and Eduardo Serrano. Internet of intelligent things: Bringing artificial intelligence into things and communication networks. In *Inter-cooperative collective intelligence: Techniques and applications*. Springer, 2014.

[14] Vincent Ballet, Xavier Renard, Jonathan Aigrain, Thibault Laugel, Pascal Frossard, and Marcin Detyniecki. Imperceptible adversarial attacks on tabular data. *arXiv:1911.03274*, 2019.

[15] Nathalie Baracaldo and James Joshi. An adaptive risk management and access control framework to mitigate insider threats. *Computers & Security*, 2013.

[16] L. Bauer, L. F Cranor, et al. Real life challenges in access-control management. In *Conference on Human Factors in Computing Systems*, 2009.

[17] András A Benczúr, Levente Kocsis, and Róbert Pálovics. Online machine learning in big data streams. *arXiv*, 2018.

[18] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009.

[19] Yoshua Bengio. *Learning deep architectures for AI*. Now Publishers Inc, 2009.

[20] Yahya Benkaouz, Mohammed Erradi, and Bernd Freisleben. Work in progress: K-nearest neighbors techniques for abac policies clustering. In *ACM International Workshop on Attribute Based Access Control*, 2016.

[21] Hugh Boyes, Bil Hallaq, Joe Cunningham, and Tim Watson. The industrial internet of things (iiot): An analysis framework. *Computers in industry*, 101:1–12, 2018.

[22] André Brandão, Ricardo Mendes, and João P Vilela. Prediction of mobile app privacy preferences with user profiles via federated learning. In *Proceedings of the Twelveth ACM Conference on Data and Application Security and Privacy*, pages 89–100, 2022.

[23] Leo Breiman. Random forests. *Machine learning*, 2001.

[24] T. Bui, S. D Stoller, and J. Li. Greedy and evolutionary algorithms for mining relationship-based access control policies. *Computers & Security*, 2019.

[25] Thang Bui and Scott D Stoller. A decision tree learning approach for mining relationship-based access control policies. In *ACM SACMAT*, 2020.

[26] Thang Bui and Scott D Stoller. Learning attribute-based and relationship-based access control policies with unknown values. In *International Conference on Information Systems Security*. Springer, 2020.

[27] Thang Bui, Scott D Stoller, and Hieu Le. Efficient and extensible policy mining for relationship-based access control. In *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies*, pages 161–172, 2019.

[28] Thang Bui, Scott D Stoller, and Jiajie Li. Mining relationship-based access control policies from incomplete and noisy data. In *International Symposium on Foundations and Practice of Security*. Springer, 2018.

[29] Thang Bui, Scott D Stoller, and Jiajie Li. Greedy and evolutionary algorithms for mining relationship-based access control policies. *Computers & Security*, 2019.

[30] Luca Cappelletti et al. On the quality of classification models for inferring abac policies from access logs. In *Big Data*. IEEE, 2019.

[31] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *S&P*. IEEE, 2017.

[32] Francesco Cartella, Orlando Anunciacao, Yuki Funabiki, Daisuke Yamaguchi, Toru Akishita, and Olivier Elshocht. Adversarial attacks for tabular data: Application to fraud detection and imbalanced data. *arXiv*, 2021.

[33] Chin-Chen Chang, Iuon-Chang Lin, and Chia-Te Liao. An access control system with time-constraint using support vector machines. *Int. J. Netw. Secur.*, 2(2):150–159, 2006.

[34] Y. Cheng, K. Bijon, and R. Sandhu. Extended rebac administrative models with cascading revocation and provenance support. In *SACMAT*. ACM, 2016.

[35] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *IEEE CVPR*, 2017.

[36] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 1995.

[37] Carlos Cotrini, Thilo Weghorn, and David Basin. Mining abac rules from sparse logs. In *Euro S&P*. IEEE, 2018.

[38] Li Da Xu, Wu He, and Shancang Li. Internet of things in industries: A survey. *IEEE Transactions on industrial informatics*, 10(4), 2014.

[39] F Dalpiaz. Requirements data sets (user stories). *Mendeley Data, v1*, 2018.

[40] Fabiano Dalpiaz, Ivor Van der Schalk, and Garm Lucassen. Pinpointing ambiguity and incompleteness in requirements engineering via information visualization and nlp. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2018.

[41] S. Das, B. Mitra, V. Atluri, J. Vaidya, and S. Sural. Policy engineering in rbac and abac. In *From Database to Cyber Security*. Springer, 2018.

[42] Maarten Decat, Jasper Bogaerts, Bert Lagaisse, and Wouter Joosen. The e-document case study: functional analysis and access control requirements. *CW Reports CW654, Department of Computer Science, KU Leuven*, 2014.

[43] Maarten Decat, Jasper Bogaerts, Bert Lagaisse, and Wouter Joosen. The workforce management case study: functional analysis and access control requirements. *CW Reports CW655, Dept. of Computer Science, KU Leuven*, 2014.

[44] Deborah D Downs, Jerzy R Rub, et al. Issues in discretionary access control. In *IEEE S& P*, 1985.

[45] Maryem Ait El Hadj et al. Abac rule reduction via similarity computation. In *ICNS*. Springer, 2017.

[46] David F Ferraiolo, Dennis M Gilbert, and Nickilyn Lynch. An examination of federal and commercial access control policy needs. In *NIST-NCSC*, 1995.

[47] Kathi Fisler, Shriram Krishnamurthi, Leo A Meyerovich, and Michael Carl Tschantz. Verification and change-impact analysis of access-control policies. In *Software Engineering*, 2005.

[48] Philip WL Fong and Ida Siahaan. Relationship-based access control policies and their policy languages. In *Proceedings of the 16th ACM symposium on Access control models and technologies*, pages 51–60, 2011.

[49] Mario Frank, David Basin, and Joachim M Buhmann. A class of probabilistic models for role engineering. In *CCS*. ACM, 2008.

[50] Mario Frank et al. A class of probabilistic models for role engineering. In *ACM CCS*, 2008.

[51] Shota Fukui, Jaehoon Yu, and Masanori Hashimoto. Distilling knowledge for non-neural networks. In *APSIPA ASC*. IEEE, 2019.

[52] Ian Goodfellow and Nicolas Papernot. The challenge of verification and testing of machine learning. *Cleverhans-blog*, 2017.

[53] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[54] Jindong Gu and Volker Tresp. Semantics for global and local interpretation of deep neural networks. *arXiv:1910.09085*, 2019.

[55] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[56] Varun Gumma, Barsha Mitra, Soumyadeep Dey, Pratik Shashikantbhai Patel, Sourabh Suman, and Saptarshi Das. Pammela: Policy administration methodology using machine learning. *arXiv preprint arXiv:2111.07060*, 2021.

[57] John T Hancock and Taghi M Khoshgoftaar. Survey on categorical data for neural networks. *Journal of Big Data*, 2020.

[58] M. A Harrison, Walter L Ruzzo, and Jeffrey D Ullman. Protection in operating systems. *Communications of the ACM*, 1976.

[59] Kaiming He et al. Deep residual learning for image recognition. In *CVPR*, 2016.

[60] John Heaps, Ram Krishnan, Yufei Huang, Jianwei Niu, and Ravi Sandhu. Access control policy generation from user stories using machine learning. In *DBSec*. Springer, 2021.

[61] John Heaps, Xiaoyin Wang, Travis Breaux, and Jianwei Niu. Toward detection of access control models from source code via word embedding. In *ACM SACMAT*, 2019.

[62] G. Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv:1503.02531*, 2015.

[63] Yaoshiang Ho and Samuel Wookey. The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling. *IEEE Access*, 8:4806–4813, 2019.

[64] V. C Hu, D. Ferraiolo, et al. Guide to attribute based access control (abac) definition and considerations (draft). *NIST special publication*.

[65] Vincent Hu. Machine learning for access control policy verification. Technical report, National Institute of Standards and Technology, 2021.

[66] Vincent C Hu, D Richard Kuhn, David F Ferraiolo, and Jeffrey Voas. Attribute-based access control. IEEE, 2015.

[67] Vincent C Hu and Rick Kuhn. Access control policy verification. *Computer*, 2016.

[68] Wenpeng Hu, Zhou Lin, Bing Liu, Chongyang Tao, Zhengwei Tao, Jinwen Ma, Dongyan Zhao, and Rui Yan. Overcoming catastrophic forgetting for continual learning via model adaptation. In *ICLR*, 2018.

[69] Gao Huang et al. Densely connected convolutional networks. In *CVPR*, 2017.

[70] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*. PMLR, 2015.

[71] Qatrunnada Ismail, Tousif Ahmed, Apu Kapadia, and Michael K Reiter. Crowdsourced exploration of security configurations. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 467–476, 2015.

[72] Padmavathi Iyer and Amirreza Masoumzadeh. Mining positive and negative attribute-based access control policy rules. In *SACMAT*. ACM, 2018.

[73] Padmavathi Iyer and Amirreza Masoumzadeh. Active learning of relationship-based access control policies. In *SACMAT*, 2020.

[74] Amani Abu Jabal, Elisa Bertino, et al. Polisma-a framework for learning attribute-based access control policies. In *ESORICS*, 2020.

[75] Sadhana Jha, Shamik Sural, Vijayalakshmi Atluri, and Jaideep Vaidya. An administrative model for collaborative management of abac systems and its security analysis. In *IEEE CIC*, 2016.

[76] L. Karimi, M. Aldairi, J. Joshi, and M. Abdelhakim. An automatic attribute based access control policy extraction from access logs. *IEEE TDSC*, 2021.

[77] Leila Karimi, Mai Abdelhakim, and James Joshi. Adaptive abac policy learning: A reinforcement learning approach. *arXiv preprint arXiv:2105.08587*, 2021.

[78] Leila Karimi and James Joshi. An unsupervised learning based approach for mining attribute based access control policies. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018.

[79] Alan H Karp, Harry Haury, and Michael H Davis. From abac to zbac: the evolution of access control models. *Journal of Information Warfare*, 2010.

[80] Branko Kavšek and Nada Lavrač. Apriori-sd: Adapting association rule learning to subgroup discovery. *Applied Artificial Intelligence*, 2006.

[81] Branko Kavšek and Nada Lavrač. APRIORI-SD: Adapting association rule learning to subgroup discovery. *Applied Artificial Intelligence*, 2006.

[82] A. Kaya et al. Analysis of transfer learning for deep neural network based plant classification models. *Computers and electronics in agriculture*, 2019.

[83] Pabitr Mohan Khilar, Vijay Chaudhari, and Rakesh Ranjan Swain. Trust-based access control in cloud computing using machine learning. In *Cloud Computing for Geospatial Big Data Analytics*. Springer, 2019.

[84] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *National Academy of Sciences*, 2017.

[85] L. Krautsevich et al. Towards attribute-based access control policy engineering using risk. In *Intl. Workshop on Risk Assessment and Risk-driven Testing*, 2013.

[86] D Richard Kuhn, Vincent Hu, David F Ferraiolo, Raghu N Kacker, and Yu Lei. Pseudo-exhaustive testing of attribute based access control rules. In *ICSTW*. IEEE, 2016.

[87] Nishant Kumar, Siddharth Vimal, Kanishka Kayathwal, and Gaurav Dhama. Evolutionary adversarial attacks on payment systems. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 813–818. IEEE, 2021.

[88] Ram Shankar Siva Kumar, Magnus Nyström, John Lambert, Andrew Marshall, Mario Goertzel, Andi Comissoneru, Matt Swann, and Sharon Xia. Adversarial machine learning-industry perspectives. In *2020 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2020.

[89] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[90] Francesco Leofante, Nina Narodytska, Luca Pulina, and Armando Tacchella. Automated verification of neural networks: Advances, challenges and perspectives. *arXiv preprint arXiv:1805.09938*, 2018.

[91] Ninghui Li, Qihua Wang, Wahbeh Qardaji, Elisa Bertino, Prathima Rao, Jorge Lobo, and Dan Lin. Access control policy combining: theory meets practice. In *Proceedings of the 14th ACM symposium on Access control models and technologies*, pages 135–144, 2009.

[92] Jialiu Lin, Bin Liu, Norman Sadeh, and Jason I Hong. Modeling {Users'} mobile app privacy preferences: Restoring usability in a sea of permission settings. In *10th Symposium On Usable Privacy and Security (SOUPS 2014)*, pages 199–212, 2014.

[93] A. Liu, X. Du, and N. Wang. Efficient access control permission decision engine based on machine learning. *Security & Communication Networks*, 2021.

[94] Bin Liu, Mads Schaarup Andersen, Florian Schaub, Hazim Almuhimedi, Shikun Aerin Zhang, Norman Sadeh, Yuvraj Agarwal, and Alessandro Acquisti. Follow my recommendations: A personalized privacy assistant for mobile app permissions. In *Twelfth symposium on usable privacy and security (SOUPS 2016)*, pages 27–41, 2016.

[95] Evan Martin and Tao Xie. Inferring access-control policy properties via machine learning. In *IEEE POLICY*, 2006.

[96] Yael Mathov, Eden Levy, Ziv Katzir, Asaf Shabtai, and Yuval Elovici. Not all datasets are born equal: On heterogeneous tabular data and adversarial examples. *Knowledge-Based Systems*, 2022.

[97] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*. Elsevier, 1989.

[98] N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan. A survey on bias and fairness in machine learning. *arXiv:1908.09635*, 2019.

[99] Ricardo Mendes, André Brandão, Joao P Vilela, and Alastair R Beresford. Effect of user expectation on mobile app privacy: A field study. In *2022 IEEE international conference on pervasive computing and communications (PerCom)*, pages 207–214. IEEE, 2022.

[100] Andrew Meneely, Ben Smith, and Laurie Williams. Appendix b: itrust electronic health care system case study. *Software and Systems Traceability*, 2012.

[101] Decebal Mocanu, Fatih Turkmen, Antonio Liotta, et al. Towards abac policy mining from logs with deep learning. In *Proceedings of the 18th International Multiconference, ser. Intelligent Systems*, 2015.

[102] I. Molloy et al. Adversaries' holy grail: access control analytics. In *Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*, 2011.

[103] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[104] Masoud Narouei, Hamed Khanpour, Hassan Takabi, Natalie Parde, and Rodney Nielsen. Towards a top-down policy engineering framework for attribute-based access control. In *ACM SACMAT*, 2017.

[105] Masoud Narouei and Hassan Takabi. A nature-inspired framework for optimal mining of attribute-based access control policies. In *ICSPCS*. Springer, 2019.

[106] Radford M Neal. Markov chain sampling methods for dirichlet process mixture models. *Journal of computational and graphical statistics*, 2000.

[107] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *CVPR*. IEEE, 2015.

[108] Qun Ni, Jorge Lobo, Seraphin Calo, Pankaj Rohatgi, and Elisa Bertino. Automating role-based provisioning by learning from examples. In *ACM SACMAT*, 2009.

[109] Mohammad Nur Nobi, Ram Krishnan, Yufei Huang, Mehrnoosh Shakarami, and Ravi Sandhu. Toward deep learning based access control. In *ACM CODASPY*, 2022.

[110] F. Pedregosa et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 2011.

[111] Bahman Rashidi, Carol Fung, and Tam Vu. Dude, ask the experts!: Android resource access permission recommendation with recdroid. In *2015 IFIP/IEEE international symposium on integrated network management (IM)*, pages 296–304. IEEE, 2015.

[112] Ramprasad Ravichandran, Michael Benisch, Patrick Gage Kelley, and Norman M Sadeh. Capturing social networking privacy preferences. In *International symposium on privacy enhancing technologies symposium*, pages 1–18. Springer, 2009.

[113] S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 1991.

[114] Zhang Sainan and Zheng Changyou. Research and application of rigorous access control mechanism in distributed objects system. In *ITNEC*. IEEE, 2019.

[115] Matthew W Sanders and Chuan Yue. Mining least privilege attribute based access control policies. In *ACSAC*, 2019.

[116] Ravi Sandhu and Qamar Munawer. The arbac99 model for administration of roles. In *IEEE ACSAC*, 1999.

[117] Ravi S. Sandhu. Lattice-based access control models. *Computer*, 1993.

[118] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. Role-based access control models. *Computer*, 1996.

[119] Ravi S Sandhu et al. Access control: principle and practice. *IEEE communications magazine*, 1994.

[120] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61, 2015.

[121] Cedric Seger. An investigation of categorical variable encoding techniques in machine learning: binary versus one-hot and feature hashing, 2018.

[122] Daniel Servos and Sylvia L Osborn. Current research and open problems in attribute-based access control. *ACM Computing Surveys (CSUR)*, 2017.

[123] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *arXiv:1705.08690*, 2017.

[124] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *ICML*. PMLR, 2017.

[125] Sara Sinclair and Sean W Smith. Preventative directions for insider threat mitigation via access control. In *Insider Attack and Cyber Security*. Springer, 2008.

[126] Emiliano Sisinni, Abusayeed Saifullah, Song Han, Ulf Jennehag, and Mikael Gidlund. Industrial internet of things: Challenges, opportunities, and directions. *IEEE transactions on industrial informatics*, 14(11), 2018.

[127] Kriti Srivastava and Narendra Shekokar. Machine learning based risk-adaptive access control system to identify genuineness of the requester. In *Modern Approaches in Machine Learning and Cognitive Science: A Walkthrough*, pages 129–143. Springer, 2020.

[128] Scott D Stoller. An administrative model for relationship-based access control. In *DBSec*. Springer, 2015.

[129] M. Sundararajan et al. Axiomatic attribution for deep networks. In *ICML*, 2017.

[130] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[131] Nima Tajbakhsh, Jae Y Shin, Suryakanth R Gurudu, R Todd Hurst, Christopher B Kendall, Michael B Gotway, and Jianming Liang. Convolutional neural networks for medical image analysis: Full training or fine tuning? *IEEE transactions on medical imaging*, 35(5), 2016.

[132] Roshan K Thomas, Ravi S Sandhu, et al. Discretionary access control in object-oriented databases: Issues and research directions. In *Proc. 16th National Computer Security Conference*, 1993.

[133] Ian P Turnipseed. *A new scada dataset for intrusion detection research*. Mississippi State University, 2015.

[134] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen. Wikipedia workload analysis for decentralized hosting. *Computer Networks*, 2009.

[135] L. V d Maaten and G. Hinton. Visualizing data using t-sne. *JMLR*, 2008.

[136] Richard Van De Stadt. Cyberchair: A web-based groupware application to facilitate the paper reviewing process. *arXiv*, 2012.

[137] Cheng G Weng and Josiah Poon. A new evaluation measure for imbalanced datasets. In *7th Australasian Data Mining Conference-Volume 87*, 2008.

[138] Chengcheng Xiang, Yudong Wu, et al. Towards continuous access control validation and forensics. In *CCS*. ACM, 2019.

[139] Xusheng Xiao, Amit Paradkar, Suresh Thummalapenta, and Tao Xie. Automated extraction of security policies from natural-language software documents. In *ACM Symposium on the Foundations of Software Engineering*, 2012.

[140] Jierui Xie, Bart Piet Knijnenburg, and Hongxia Jin. Location sharing privacy preference: analysis and personalized recommendation. In *Proceedings of the 19th international conference on Intelligent User Interfaces*, pages 189–198, 2014.

[141] Han Xu, Yao Ma, Hao-Chen Liu, Debayan Deb, Hui Liu, Ji-Liang Tang, and Anil K Jain. Adversarial attacks and defenses in images, graphs and text: A review. *International Journal of Automation and Computing*, 17(2):151–178, 2020.

[142] Tianyin Xu, Han Min Naing, Le Lu, and Yuanyuan Zhou. How do system administrators resolve access-denied issues in the real world? In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2017.

[143] Zhongyuan Xu and Scott D Stoller. Mining attribute-based access control policies. *TDSC*, 2014.

[144] Jaehong Yoon, Eunho Yang, Jeongtae Lee, and Sung Ju Hwang. Lifelong learning with dynamically expandable networks. *arXiv:1708.01547*, 2017.

[145] Jinsung Yoon, Lydia N Drumright, and Mihaela Van Der Schaar. Anonymization through data synthesis using generative adversarial networks (ads-gan). *IEEE journal of biomedical and health informatics*, 24(8):2378–2388, 2020.

[146] Yiqun Zhang and Yiu-ming Cheung. An ordinal data clustering algorithm with automated distance learning. In *AAAI Conference on Artificial Intelligence*, 2020.

[147] Lu Zhou, Chunhua Su, Zhen Li, Zhe Liu, and Gerhard P Hancke. Automatic fine-grained access control in scada by machine learning. *Future Generation Computer Systems*, 2019.

# VITA

Mohammad Nur Nobi is a Ph.D. candidate in the Department of Computer Science at the University of Texas at San Antonio (UTSA). He was born and raised in Noakhali, Bangladesh. He earned his Bachelor in Computer Science and Engineering (CSE) in 2011 from Mawlana Bhashani Science and Technology University (MBSTU), Bangladesh. He joined Samsung R&D Institute Bangladesh in 2011 and worked for several years. In parallel, he completed his master's degree in CSE from MBSTU, Bangladesh, in 2015. He joined Bangladesh Atomic Energy Commission (BAEC) as a Scientific Officer at the Institue of Computer Science (ICS) in 2016. He continued there until he moved to UTSA to pursue the doctoral program in Fall 2018. Mohammad Nur Nobi worked under the supervision of Professor Ravi Sandhu and Professor Ram Krishnan at the Institute for Cyber Security (ICS). His areas of research include cybersecurity, machine learning, and data analytics. In particular, he focused on utilizing data analysis and machine learning techniques to develop an automated access control system. His research has been published at prominent computer security conferences such as ESORICS, CODASPY, etc. In addition, he serves as a program committee member and reviewer for different workshops, conferences, and journals. He is an active member of ACM and IEEE.