IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 25 February 2024, accepted 25 March 2024, date of publication 3 April 2024, date of current version 23 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3384460

METHODS

Machine Learning With Computer Networks: Techniques, Datasets, and Models

HAITHAM AFIFI[1], (Member, IEEE), SABRINA POCHABA[2], ANDREAS BOLTRES[3], DOMINIC LANIEWSKI[4], (Graduate Student Member, IEEE), JANEK HABERER[5], LEONARD PAELEKE[6,7], REZA POORZARE[8], (Member, IEEE), DANIEL STOLPMANN[9], NIKOLAS WEHNER[10], ADRIAN REDDER[11], ERIC SAMIKWA[12], AND MICHAEL SEUFERT[13], (Senior Member, IEEE)

[1]Accenture, 61476 Kronberg im Taunus, Germany
[2]Salzburg Research Forschungsgesellschaft m.b.H., 5020 Salzburg, Austria
[3]Autonomous Learning Robots Laboratory, Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany
[4]Institute of Computer Science, Osnabrück University, 49076 Osnabrück, Germany
[5]Distributed Systems Group, Kiel University, 24118 Kiel, Germany
[6]Digital Engineering Faculty, University of Potsdam, 14482 Potsdam, Germany
[7]Digital Health & Machine Learning, Hasso Plattner Institute, 14482 Potsdam, Germany
[8]Wirtschaft Center of Applied Research, Data-Centric Software Systems (DSS) Research Group, Institute of Applied Research, Hochschule Karlsruhe Technik, 76133 Karlsruhe, Germany
[9]Institute of Communication Networks, Hamburg University of Technology, 21073 Hamburg, Germany
[10]Chair of Communication Networks, University of Würzburg, 97074 Würzburg, Germany
[11]Universität Paderborn, 33098 Paderborn, Germany
[12]Institute of Computer Science, University of Bern, 3012 Bern, Switzerland
[13]Chair of Networked Embedded Systems and Communication Systems, University of Augsburg, 86159 Augsburg, Germany

Corresponding author: Michael Seufert (michael.seufert@uni-a.de)

This work was supported by German Research Foundation [Deutsche Forschungsgemeinschaft (DFG)] under Grant SE 3163/3-1, project number: 500105691 (UserNet). This work was also supported by the Federal Ministry of Education and Research of Germany under Grant 16KISK011 (Open6GHub) as well as by the Federal Ministry for Economic Affairs and Climate Action of Germany under Grant 68GX21002 (Marispace-X).

ABSTRACT Machine learning has found many applications in network contexts. These include solving optimisation problems and managing network operations. Conversely, networks are essential for facilitating machine learning training and inference, whether performed centrally or in a distributed fashion. To conduct rigorous research in this area, researchers must have a comprehensive understanding of fundamental techniques, specific frameworks, and access to relevant datasets. Additionally, access to training data can serve as a benchmark or a springboard for further investigation. All these techniques are summarized in this article; serving as a primer paper and hopefully providing an efficient start for anybody doing research regarding machine learning for networks or using networks for machine learning.

INDEX TERMS Computer networking, datasets, machine learning, metrics, tools.

I. INTRODUCTION

In recent years, the ever-growing interconnection of businesses and people and their increased reliance on networked services has prompted computer network architectures to continually grow in size and complexity. Moreover, with the increased efficiency and convenience of network-based services and businesses, the expectations of enterprises and people with respect to network performance indicators such as latency, throughput, reliability and resilience are steadily growing. Consequently, conventional algorithmic and heuristic-based approaches for network management tasks are starting to fall behind the expected levels of performance, as they fail to deliver timely and nuanced decisions in the face of the complex environment they are operating in. Meanwhile, Machine Learning (ML) has shown remarkable results in various problem domains such as discovering new antibiotic drugs [1], generating high-fidelity images from arbitrary text prompts [2] and even finding new mathematical conjectures [3]. Such successes usually become very visible even beyond the research community, and thus ML has soared in popularity in the past few years. Generally, Artificial

The associate editor coordinating the review of this manuscript and approving it for publication was Kaigui Bian.

Intelligence (AI) refers to machines or systems that can perform tasks typically requiring human intelligence, such as learning, reasoning, problem-solving, perception, language understanding, and decision-making. ML is a subfield of AI that concentrates on developing algorithms and statistical models. These models enable computers to perform tasks without explicit programming. In other words, it involves using statistical techniques to enable machines to learn from data and improve their performance over time. ML models repeatedly show their potential for delivering high-quality output (e.g. classifications/decisions, regression values and generated artifacts) in highly complex environments with non-trivial decision boundaries. Generally, for that sort of environment, the proposed ML model greatly reduces the compute resources needed to generate an adequate response and/or generates outputs that are much ''better'' than what existing models could deliver. That being said, for more complex problem domains, most ML approaches require substantial amounts of compute resources and training data. Since most ML models aim at generalizing from specific records of data, the quality of these data samples is essential to the overall model performance. This often means that large amounts of data records are required to depict a sufficiently representative portion of the problem's data domain. Also, more sophisticated models can quickly explode in terms of parameter/compute operation count and thus often require specialized training hardware (i.e. memory and compute). Nevertheless, the continuous improvement of used hardware as well as the increased attention towards training data acquisition, preparation and generation has paved the way for ML to enter into more and more application domains.

Computer networking is a highly complex problem domain with a plethora of tasks and problems that, to this day, are solved predominantly through hand-crafted, algorithmic, or heuristic methods. These methods have to respect a wide range of topologies, network types and scopes, configurations, hardware and protocol stacks, traffic patterns, and other sources of variation. Furthermore, there are many different ways to assess network performance, and in many cases, minimum performance guarantees and security policies add special constraints to the optimization problem. Additionally, contemporary networks use specialized hardware to deliver optimized performance, e.g. for forwarding packets at line speed. Oftentimes, this hardware does not easily allow ML models to replace existing functionality, e.g. because certain types of computations are not supported or because the storage is not available for more complex ML models. Finally, while network administrators and networking researchers do monitor their networks in action, the amount of useful ML training data in networking – data that is not noisy nor incomplete, publicly available, and diverse enough to cover large parts of the problem's underlying data domain – is only a fraction of what other problem domains have at their disposal. As a consequence, optimizing network performance has so far been largely beyond the reach of ML research. However, given the increased visibility of ML, researchers are beginning to take on the aforementioned challenges of the networking community on ML, and combining ML and networking in research seems more attractive than ever. Furthermore, computer network infrastructures have been used recently to improve the performance of existing ML approaches, e.g. by distributing the training process or the data collection to improve resource utilization or training speed.

ML is a very active and rapidly expanding research field that includes an abundance of learning techniques, model types, tools and frameworks, practices, and application possibilities. Although we focus here on ML models, some applications require considering the whole running system, i.e., AI system, to properly evaluate and understand the output, instead of focusing solely on the ML models [4]. This paper is intended as a primer/practical guide for researchers who are keen on quickly applying ML to problems in computer networking and/or leveraging networking techniques to improve the performance of their ML systems but feel overwhelmed by the possibilities the intersection of ML and computer networking provides. The key points of the paper are the following:

- It first introduces the most relevant concepts and model architectures of ML and then puts them into the context of the different networking problem domains and the latest advancements therein,
- It exposes the currently open problems within computer networking and introduces a selection of different tools, data sets, and approaches that have been popular among the research community and might serve as a starting point for future work,
- It covers several techniques for utilizing networks to improve ML efficiency, such as reducing resource requirements via Split Learning (SL) and distributed training via Federated Learning (FL) or incorporating the right inductive biases into ML models to improve their ability to generalize from limited data,
- It discusses challenges related to networks for ML, such as resource constraints, security concerns, and the lack of understanding of how ML models make decisions (and how techniques such as Explainable Artificial Intelligence (XAI) may help in gaining understanding),
- It comprehensively provides pointers for further study on related surveys and research.

The organization of the paper is visualized in Figure 1, and the remainder is organized as follows: Section II explains the basic concepts and categories of ML and relates common networking problems to them. Section III introduces the ML subfield of deep learning, which has been responsible for most of the recent ML breakthroughs, elaborating on the most common model architectures and how and why they are suited to specific tasks within computer networking. Thereafter, Section IV sheds light on the variety of accessible data sets, tools, and frameworks that ease the development and training of ML-powered networking systems. Section V discusses explainability in Artificial Intelligence (XAI),

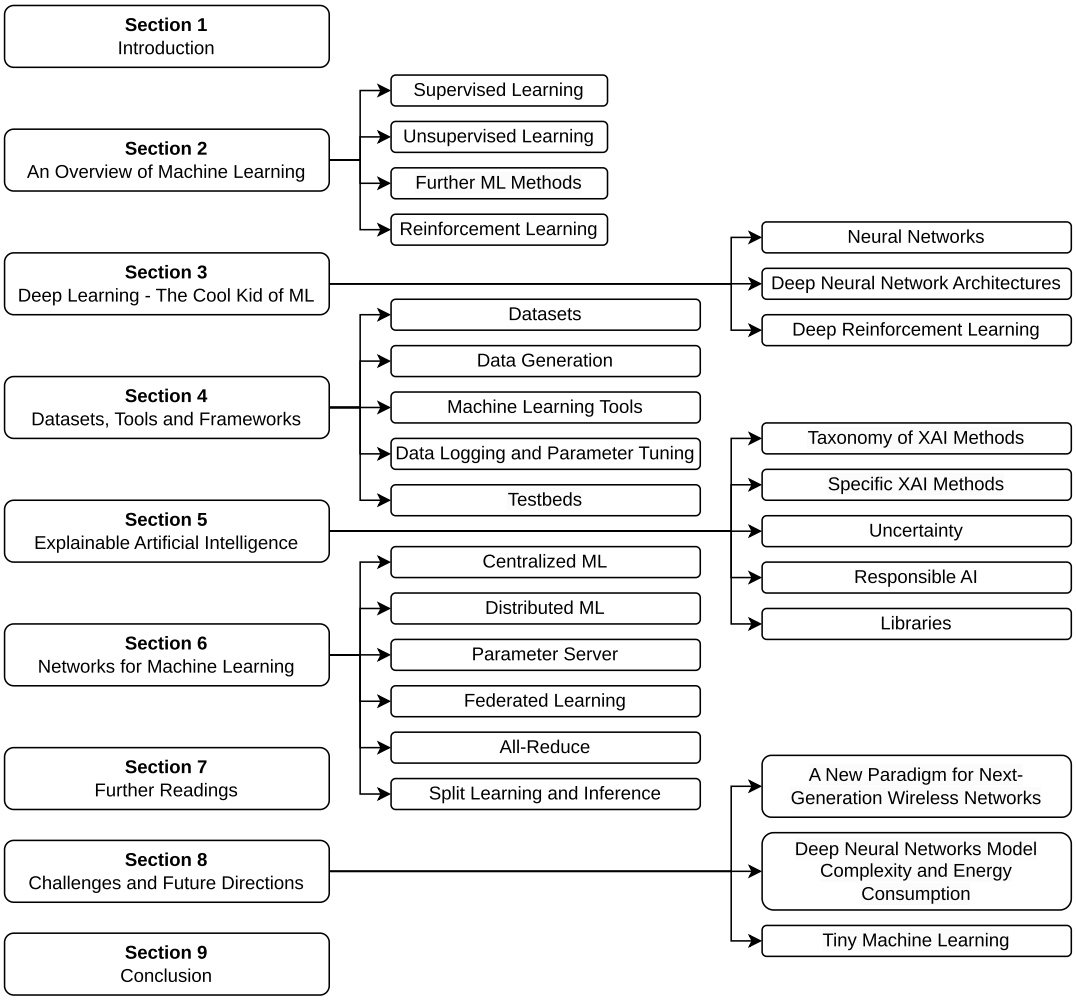


FIGURE 1. Overview of the organization of this paper.

which is rightfully gaining traction because many recently tapped application domains (including computer networks) come with amounts of complexity and risk that disqualify fully black-box ML models for widespread adoption. Section VI broadens the scope presented up until now and introduces ML techniques and paradigms such as distributed and parallel learning. These techniques leverage existing networking concepts and technology and seem useful, if not mandatory, for many problems in the networking domain. Section VII and Section VIII give an overview of related survey papers and open challenges in the concerned areas, and finally, Section IX concludes this paper by summarizing the content presented in this paper and providing perspectives on the open challenges and questions of ML in networking and vice versa.

II. AN OVERVIEW OF MACHINE LEARNING

AI is the discipline of machines that solve problems by perceiving the environment and using some form of knowledge model in order to derive solutions and conclusions. ML is an integral part of AI, of which it is considered a major subfield [5]. ML models are statistically and computationally derived from evidence in the form of historical data or experience instead of explicitly programming a machine for a task. The three traditional ML paradigms are supervised, unsupervised, and Reinforcement Learning (RL). Methods can be categorized into these paradigms by the type of feedback the learning system receives. In supervised learning, exact feedback is available in the form of data labels. In unsupervised learning, on the other hand, data is only partially labeled or completely unlabeled. Finally, in RL, implicit feedback is available for observed data in terms of a so-called reward function that labels data by a numerical value. We will now discuss the three main ML paradigms with a focus on the most popular ones. We then briefly touch on some additional branches of ML that are relevant to computer networking.

A. SUPERVISED LEARNING

Supervised learning is the first of the three main types of ML and encompasses models that predict target values y_i for given data points x_i. The starting point for the learning

problem is a *data set* that consists of input-output data points $D = (x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$. The goal is to learn a function h mapping from the input domain to the target domain such that $\hat{y}_i = h(x_i)$ for all data points. Both input and output domains can take various shapes, such as boolean or scalar values, euclidean vectors or more complex representations such as graphs. Depending on the type of output domain, supervised learning is generally divided into *classification* and *regression* problems. Examples of popular network applications that use supervised learning are traffic prediction [6] and classifying security attacks [7].

1) CLASSIFICATION

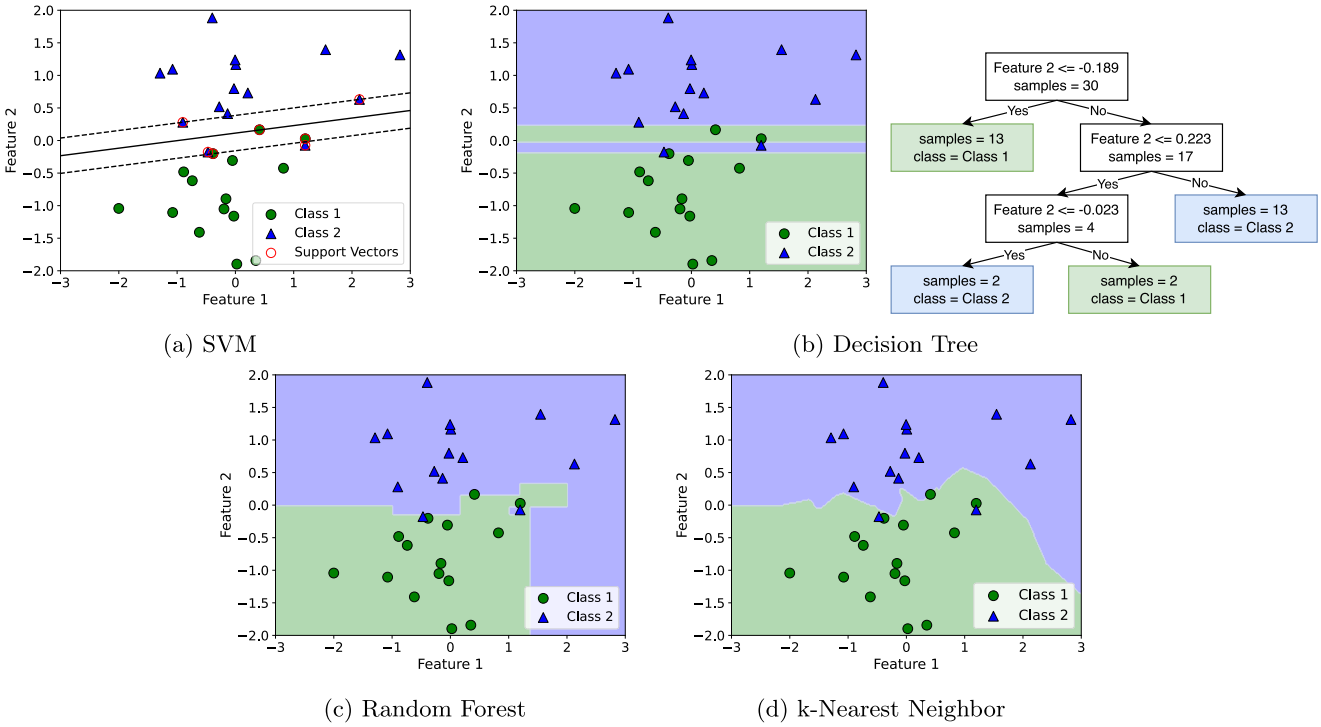
In classification problems, the output domain is finite, e.g. true/false, sunny/cloudy/rainy or the set of digits 0-9. Examples from the networking domain include anomaly detection [8] (''Given the current network monitoring data, is the network showing abnormal behavior?'') and failure prediction [9] (''Is this network node going to fail?''). The most fundamental models for classification are explained in the following paragraphs.

- *Support Vector Machines (SVMs)* [10] aim at constructing a so-called *maximum margin separator* - a decision boundary that divides samples of two different classes with a maximum possible distance to the boundary. This situation is depicted in Figure 2a. The solid black line represents the maximum margin separator and the two dashed lines visualize the margins to both classes. The nearest data points to the separator are called the support vectors (red circles), as they support the position of the decision boundary. Generally, the larger the margin, the better the generalization of the model, as it reduces the risk of misclassifying new, unseen data. Since the decision boundary is a separating hyperplane, the classification task fails for data that is not linearly separable. However, SVMs can also be used for non-linearly separable data by applying the *kernel trick*. It transforms the data into a higher-dimensional space where it becomes linearly separable and a separating hyperplane is calculated. When that linear hyperplane is transformed back into the original space, it becomes a non-linear or even incoherent hypersurface.
- *Decision Trees* [11] are structured like an inverted tree, with a root node at the top, branching out into internal nodes, and ending in leaf nodes at the bottom. The data is split at the root node and the internal nodes based on a threshold value for a feature. The splitting process continues until a stopping criterion is met, such as reaching a maximum depth or minimum number of samples in a leaf node. Leaf nodes represent the final predictions of the decision tree. The majority class in each leaf node is used as the prediction. Figure 2b visualizes a simple example decision tree (right side) with a root node and two leaf nodes for the same data as in the SVM example (left side). Data points where $Feature2 \leq -0.103$ are assigned *Class 1*, all others are assigned *Class 2*. This decision boundary is visualized as the color step from green to blue in the left plot. Decision Trees are a simple yet powerful tool to reach conclusions from input data with a high degree of human explainability (see Section V).
- *Random Forests* [12] create a collection of decision trees, each trained on a different subset of the training data. To achieve as little inter-tree correlation as possible, a random subset of features is considered for each split at each node when constructing the decision trees. The results of all individual decision trees are aggregated to a final prediction. The class with the majority vote among the trees is chosen as the final prediction. Figure 2c visualizes the decision boundary of a random forest with three individual trees for the same data set used for the SVM and decision tree examples. Compared to single decision trees, random forests are known to improve the prediction accuracy as well as to reduce overfitting [12].
- The *k-Nearest Neigbors (KNN)* [13] algorithm is a simple technique to assign class labels to new data points by examining the class labels of its k-nearest neighbors with known labels. Given the features of the input data point, these k-nearest neighbors are determined by calculating a distance metric in the input space, e.g., the Euclidean distance, Manhattan distance, or Minkowski distance. The class label of the majority of these neighbors is then inherited for the new data point. Figure 2d visualizes the decision boundary for the known data set using the Minkowski distance and $k = 5$.

2) REGRESSION

In regression problems, the output domain is continuous, e.g., $\mathbb{R}^n$ ($n \geq 1$). Examples from the networking domain include network performance prediction [14] (''How will the network perform in the future, given certain network conditions and traffic?'') and traffic prediction [15] (''How much / which type of traffic will be generated in the near future?''). In principle, any function f_w with learnable parameters w can serve as a regression model. However, the structure of f_w and the optimization procedure used to update the learnable parameters are crucial to finding good function parameters efficiently. The most fundamental regression methods will be explained in the following paragraphs. All of the aforementioned classification methods can also be used for regression with slight modifications.

- *Support Vector Regression (SVR)* [16] is an extension of SVMs for regression tasks. It aims to find a function f that approximates the relationship between input features and continuous target values with a certain degree of error tolerance. The error tolerance (ϵ) defines an ϵ-tube around f. Inside this tube, errors from the regression model are not penalized. The algorithm



(a) SVM (b) Decision Tree

(c) Random Forest (d) k-Nearest Neighbor

FIGURE 2. Visualization of different classification methods based on a data set containing 30 samples and two features. The samples contain 15 samples of two different classes that are perfectly linearly separable. A Gaussian noise with a mean of 0 and a standard deviation of 0.8 is added, making classification errors likely.

maximizes the number of training data points inside this tube. ϵ is a parameter defined by the user. Similar to SVMs, the *kernel trick* can be applied to create non-linear SVRs.

- *Decision Trees* [11] can be used for regression tasks by using the average value of the samples in each leaf node as the prediction value.
- *Random Forests* [12] for regression use the average of the individual trees' predictions as the final prediction value.
- The *KNN* [13] method for regression calculates the label for the new data point by calculating the average target value of its *k*-nearest neighbors.
- The most popular regression method is *least-squares* fitting, in which the model is updated to minimize the squared L2 norms of the difference between the predicted values and their associated labels. This is known as the Mean Squared Error (MSE).

  In linear regression, this line is represented by a linear function, while in logarithmic regression, it is represented by a logarithmic function. In other words, least-squares method fits a line to the data points in a way that minimizes the sum of the squared vertical distances between the line and the points.

B. UNSUPERVISED LEARNING

As opposed to supervised learning, in unsupervised learning, the data comes without output/target values. Consequently, ML models are tasked with finding the underlying regularities in the data domain by inferring them from the given training data. The two main types of unsupervised learning, namely *clustering* and *dimensionality reduction*, differ in their use case. Supervised learning has been used for tasks such as anomaly detection, intrusion detection [17] and data traffic analyses [18].

1) CLUSTERING

Clustering approaches use the data points' feature values to find regularities in the data domain and thus divide them into multiple semantically meaningful categories. Clustering approaches such as k-means or Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [19] differ in the way cluster affiliation is calculated, for example, through data density or neighbor connectivity via measurable distance between the data points. Within the networking domain, data grouping can serve as a useful starting point for further analysis and action in a variety of problem settings, such as anomaly detection and resolution [20], task classification for scheduling [21], or traffic characterization for traffic engineering [22].

In general, there are different metrics to evaluate the performance of ML algorithms. Table 3 shows the most common metrics, which appear in the literature, used for supervised learning (with an emphasis on classification metrics that are typically used for evaluating traffic prediction) and unsupervised learning (with an emphasis on clustering

TABLE 1. Summary of supervised learning.

Model	Type	Description	Advantages	Disadvantages
Support Vector Machine (SVM)	Regression and classification	A supervised learning algorithm that finds a hyperplane that separates the data into different classes or predicts a continuous value based on the input features.	Can handle high-dimensional data, nonlinear relationships, and outliers. Can use different kernels to customize the model.	Can be computationally expensive, sensitive to hyperparameters, and difficult to interpret.
Support Vector Regression (SVR)	Regression	A type of SVM that predicts a continuous value based on the input features. It uses an epsilon-insensitive loss function to measure the error between the predicted and actual values.	Can handle high-dimensional data, nonlinear relationships, and outliers. Can use different kernels to customize the model.	Can be computationally expensive, sensitive to hyperparameters, and difficult to interpret.
Decision Trees	Regression and classification	A supervised learning algorithm that splits the data into smaller subsets based on some criteria until the leaf nodes are reached. The leaf nodes represent the class labels or the predicted values.	Easy to understand, interpret, and visualize. Can handle both numerical and categorical data. Can capture complex nonlinear relationships.	Prone to overfitting, instability, and bias. Sensitive to noise and missing values.
Random Forests	Regression and classification	An ensemble learning algorithm that combines multiple decision trees and aggregates their predictions using majority voting or averaging. It uses bootstrap sampling and feature selection to introduce randomness and reduce correlation among the trees.	Can improve the accuracy and robustness of decision trees. Can handle both numerical and categorical data. Can capture complex nonlinear relationships. Can estimate feature importance.	Prone to overfitting, especially with noisy data. Can be computationally expensive and slow to train and test. Less interpretable than single decision trees.
k-Nearest Neigbors (KNN)	Regression and classification	A lazy learning algorithm that predicts the class label or the value of a new instance based on the similarity with its k nearest neighbors in the training data. It uses a distance metric such as Euclidean or Manhattan to measure the similarity.	Simple and intuitive to implement and understand. No training required. Can handle both numerical and categorical data. Can adapt to new data dynamically.	Sensitive to noise, outliers, and irrelevant features. Can be computationally expensive and slow to test. Requires storage of the entire training data. Difficult to choose the optimal value of k.
Least Squares	Regression	A method of fitting a linear model to the data by minimizing the sum of squared errors between the observed and predicted values. It can be solved analytically using normal equations or iteratively using gradient descent or other algorithms.	Simple and fast to implement and solve. Provides a closed-form solution for linear models. Can handle multiple features and multicollinearity.	Sensitive to noise, outliers, and nonlinearity. Prone to overfitting or underfitting, depending on the complexity of the model. May suffer from numerical instability or singularity issues.

metrics as seen in intrusion detection as well as node(s) selection for data collection).

2) DIMENSIONALITY REDUCTION

This type of learning analyzes the statistical properties of the data in order to reduce the number of dimensions that sufficiently describes the data. This is particularly useful when dealing with more complex learning problems, as theoretical results show that the amount of data points needed to learn an accurate model scales exponentially with the dimensionality of the input data domain [23] (this phenomenon has been coined the ''curse of dimensionality''). While approaches like Decision Trees or Random Forest can reduce the dimensionality of the relevant portions of the data by considering only the most meaningful features, approaches like Principal Component Analysis (PCA) [24] find a reduced-cardinality combination of new features. Like clustering, this type of unsupervised learning is beneficial as a preparative step before further analysis or model training, especially since, in many real-world scenarios, it has been observed that the given data lies on manifolds of much lower dimensionality than the actual input space (the presumed general rule for this is called the *manifold hypothesis* [25]).

Table 1 summarizes supervised methods, while Table 2 summarizes unsupervised methods. Further details can be found in [26] and [27]. Regardless which method is used, it is important to watch out for over- and underfitting.

Overfitting is a condition where a statistical model begins to describe the random error in the data rather than the relationships between variables. This problem occurs when the model is too complex.

Underfitting, on the other hand, is the inverse of overfitting. It means that the statistical or ML model is too simplistic to accurately capture the patterns in the data. A sign of underfitting is that there is a high bias and low variance detected in the current model used.

C. FURTHER ML METHODS

There are various other branches of ML that are of use in computer networks, see [28] and [29]. Here, we discuss two additional ML frameworks that are presumably relevant in the networking domain.

1) PROBABILISTIC ML

Oftentimes, neither all relevant information is known or attainable prior to making a decision, nor is the environment that reacts to the taken decision purely deterministic [5]. Uncertainty may exist in the input data, in the decision model parameters and output values and even in the architecture of the decision model itself [30]. In all of these cases, probability theory provides a unified framework to cope by using probability distributions to model uncertain quantities. This framework is in principle, applicable to all ML learning paradigms, model architectures and problem domains that come with some notion of uncertainty. Since its comprehensive introduction would exceed this paper's scope, we point the interested reader to [30] for a high-level overview, and [31] for an extended overview of the core concepts of probabilistic ML.

2) HYBRID LEARNING APPROACHES

Many ML contributions do not fully fall into one of the aforementioned learning paradigms but rather combine their ideas and create new sources for learning signals. Some of these ''hybrid'' learning approaches are popular enough to earn their own description. In *semi-supervised learning*, typically, only parts of the training data are labeled [27]. To train a model in a supervised or unsupervised manner, the auxiliary information is extracted by respectively using the other learning type. *Self-supervised learning*, on the other hand, tackles shortcomings of supervised learning approaches (i.e., the need for large amounts of data and vulnerability to adversarial inputs) by using parts or representations of the input data as labels [32]. For example, in [33], a model is trained to predict future video frames by only feeding it the first few frames of a video and using the remaining frames as ''comparison'' labels.

D. REINFORCEMENT LEARNING

In the spectrum of traditional learning paradigms for intelligent agents, Reinforcement Learning (RL) is located between the two extreme domains of fully supervised and unsupervised learning. RL is particularly suitable for decision, control, and optimization problems where data and observations are received sequentially [34]. As such, RL can be applied to various challenging problems in network science [35], [36], [37]. Especially, Deep RL (DRL) methods as to be discussed in Section III-C have seen tremendous success in solving resource allocation problems in computer networking [38].

The implementation of RL is based on an RL agent that receives performance feedback called rewards as the agent interacts with an environment over time [39]. The algorithm designer typically crafts the reward as a function of the agent's sequential observations. The rewards, however, do not provide exact instructive feedback on how to change the agent's behavior, thence RL is placed in the spectrum of learning paradigms. In this section, we will describe the basics of RL and the most fundamental algorithms. Throughout, we will directly refer to applications in computer networking for almost all mentioned algorithms.

The interaction of an RL agent with its environment is described by a Markov Decision Process (MDP) as illustrated in Figure 3. Whenever one seeks to solve a problem using RL, the first step (arguably the most important) is to define the problem as an MDP. Based on this MDP one then chooses or designs a suitable RL algorithm to find a solution to the MDP. In general, an MDP can be considered as a system that can assume states s from a state space $\mathcal{S}$. The MDP transitions to a new state s' according to a controlled transition probability

TABLE 2. Summary of unsupervised learning.

Method	Task	Example	Advantage	Disadvantage
K-means clustering	Exclusive clustering	Market segmentation, document clustering, image segmentation	Simple and fast, easy to interpret	Sensitive to outliers and initial centroids, fixed number of clusters
Fuzzy k-means clustering	Overlapping clustering	Customer loyalty analysis, medical diagnosis	Allows for uncertainty and ambiguity, more flexible than hard clustering	More computationally expensive, harder to interpret
Hierarchical clustering	Agglomerative or divisive clustering	Phylogenetic analysis, social network analysis	No need to specify number of clusters, produces a dendrogram that shows the hierarchy of clusters	More computationally expensive, sensitive to noise and outliers
Principal component analysis	Dimensionality reduction	Data compression, feature extraction	Reduces complexity and noise, preserves most of the variance	Loses some information, may not be optimal for some tasks
Association rule mining	Finding frequent patterns or rules	Market basket analysis, web usage mining	Reveals interesting and useful relationships in data, can handle large datasets	May generate too many or too few rules, may need domain knowledge to evaluate rules
Autoencoder	Generative model	Image denoising, anomaly detection	Can learn complex and nonlinear representations of data, can generate new data similar to the input data	May overfit or underfit the data, may need careful tuning of parameters

TABLE 3. Common metrics of ML.

Metric	Definition	Supervised	Unsupervised
Accuracy	for classification tasks, it measures the proportion of correct predictions made by the model.	✓	
F1-score	for classification tasks, it is a balance between precision and recall.	✓	
AUC-ROC	for classification tasks, it measures the trade-off between the true positive rate and false positive rate.	✓	
Root Mean Squared Error (RMSE1)	for regression tasks, it measures the difference between predicted and actual values.	✓	
Normalized Mutual Information (NMI)	for clustering tasks, it measures the similarity between two partitions of a network.		✓
Jaccard Similarity	for community detection tasks, it measures the similarity between two sets of nodes.		✓

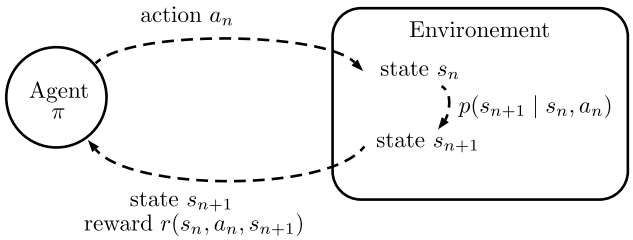


FIGURE 3. Illustration of the Markov Decision Process (MDP) feedback loop.

distribution $p(s' \mid s, a)$ after the agent has taken an action a from an action space $\mathcal{A}$. Once the system transitions to the new state, the agent receives a feedback/reward signal $r(s, a, s')$. An MDP is therefore typically stated as a four-tuple $(\mathcal{S}, \mathcal{A}, p, r)$. The design of the reward signal r is arguably the most important part of defining an MDP. We will discuss some best practices for reward design in Section III-C. Excellent introductory material for RL is the course by David Silver.[1] For background on partially observable MDPs, see the web page of Anthony R. Cassandra.[2]

The goal in reinforcement learning is to find a policy (decision rule) $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maps states to actions so as to optimize an objective function. Since all future states can potentially be influenced by an action at a current time step via the transition probabilities $p(s' \mid s, a)$, it is natural to consider an objective function that captures the whole trajectory of future rewards. The most common objective is the discounted infinite horizon accumulated reward $R := \sum_{n=1}^{\infty} \gamma^{n-1} r(s_n, a_n, s_{n+1})$ with discount factor $\gamma \in (0, 1)$. The intuition is that higher weight is given to rewards in

[1] https://www.deepmind.com/learning-resources/introduction-to-reinforcement-learning-with-david-silver

[2] http://www.pomdp.org

the near future, and the weights of future rewards decay geometrically. For background on average and total cost MDPs see [40, Chapter 4 & 5].

Given a policy π, the associated action-value function (also called Q-function) is defined as

$$Q^{\pi}(s,a) := \mathbb{E}_{\pi}\left[R \mid s_1 = s, a_1 = a\right]. \tag{1}$$

The Q-function is a fundamental object in RL and describes what accumulated reward R one can expect if we are in state s, take action a, and follow the policy π for all future states. Furthermore, for finite action spaces, the Q-function can directly be used to implement a policy by setting $\pi(s) = \text{argmax}_{a\in\mathcal{A}} Q(s,a)$. This makes RL algorithms that seek to find or approximate the optimal Q-function[3] attractive since they immediately lead to simple, implementable policies.

1) BASIC RL ALGORITHMS

RL algorithms can be roughly divided into three groups, *value-based* methods, *policy-based* methods, and *actor-critic* methods. Value-based methods seek to find or approximate value functions like the Q-function. Policy-based methods instead seek to optimize a policy π directly. Value-based methods, therefore, yield an implicit policy, whereas policy-based methods yield an explicit policy. Actor-critic methods combine learning in value- and policy-space and use a learned value function to "guide" the training of an explicit policy. See [41, p. 36] for an illustration of the actor-critic feedback loop.

Before stating some examples of popular RL algorithms, we have to distinguish some typical MDP and RL settings:

1) Continuous state (e.g. $\mathcal{S} = \mathbb{R}^n$) vs. finite state MDPs (e.g. $\mathcal{S} = \{1,\ldots,d\}$).
2) Continuous action vs. finite action MDPs.
3) Model-based vs. model-free RL problem.

Model-based RL usually focuses on offline planning of value functions and policies, where either the transition function p is given or where p will be approximated [42]. Model-free RL methods instead seek to determine what action to take in a given state without knowledge of p, e.g., solely by observing MDP transitions (s,a,r,s').

The traditional algorithms for model-based RL in finite state and action MDPs are value and policy iteration [40, Chapter 2]. Some recent applications of modern value iteration algorithms in the context of networking are age-of-information minimization in wireless broadcast networks [43] and multi-agent routing [44]. The most well-known model-free RL algorithm is tabular (simulation-based) Q-Learning, which seeks to find the optimal Q-function. Under simple conditions, Q-Learning is guaranteed to converge to the optimal Q-function if all states and actions are *explored* infinitely often [40, Section 6.6.1]. Q-Learning has been successfully applied to various problems in computer networks, e.g., network self-organization [45], network slicing [46] or virtual network embedding [47].

[3]The optimal Q-function is given by the solution to Bellman's equation [41, Section 5.6].

2) RL WITH FUNCTION APPROXIMATION

The rise of RL as a powerful tool for decision-making is largely due to the effective use of function approximation. When the state $\mathcal{S}$ becomes large or continuous, the traditional algorithms become impractical. Function approximation solves this problem by enabling RL agents to infer information about unseen state-action pairs from observed state-action pairs. The approximation may be used in policy or value space or in both, policy and value space. For example, a Q-function $Q(s,a)$ can be approximated by a function $Q^{\theta}(s,a)$ with parameters θ. This is the basis of deep Q-learning to be explained in Section III-C. Traditionally, function approximation was an important part of RL even before the rise of deep neural networks [48].

In Section III-C, we will discuss RL with deep neural networks as function approximators. Here, we highlight the traditional class of stochastic policy gradient algorithms with policy function approximation for MDPs with finite action space. Define a stochastic policy $\pi_{\theta}(s,a)$ with parameters θ; $\pi_{\theta}(s,a)$ maps states to a distribution on $\mathcal{A}$.

The stochastic policy gradient theorem [49] has given rise to a large class of algorithms, where $Q^{\pi_{\theta}}(s,a)$ is replaced by a suitable estimator. E.g., the REINFORCE algorithm [50] uses a Monte-Carlo estimator, actor-critic algorithms now add approximation in value space with an additional function approximator $Q^{w}(s,a)$ with parameters w in place of $Q^{\pi_{\theta}}(s,a)$ [51], the famous Advantage-Actor-Critic (A2C) algorithm [52] uses an approximation $A^{w}(s,a)$ of the advantage function $A^{\pi_{\theta}}(s,a) := Q^{\pi_{\theta}}(s,a) - V^{\pi_{\theta}}(s)$, where $V^{\pi_{\theta}}(s) := \mathbb{E}_{a\sim\pi_{\theta}}\left[Q^{\pi_{\theta}}(s,a)\right]$. These algorithms have been used for various scheduling and resource allocation tasks in data centers [53], wireless networks [54], edge computing [55] or vehicular networks [56].

3) EXPLORATION AND CURIOSITY IN RL

RL methods trade-off exploration vs. exploitation during training, i.e., agents either explore some random action or exploit their current best guess of the optimal action for the current state. On the other hand, some methods purely focus on exploration as a metric for learning. Such methods may seek to explore as many unseen states during training as possible. Another approach is to explore promising states, e.g., those parts of the state space where the current approximation of a certain value function is particularly bad. Such methods are known as curiosity-driven RL [57], [58].

4) MULTI-AGENT RL

Multi-Agent RL (MARL) problems are formulated as Markov games [59], where depending on the local reward structure, the agents may cooperate or compete. An illustration is given in Figure 4 from the perspective of some agent i in a Markov game environment. Most notably,

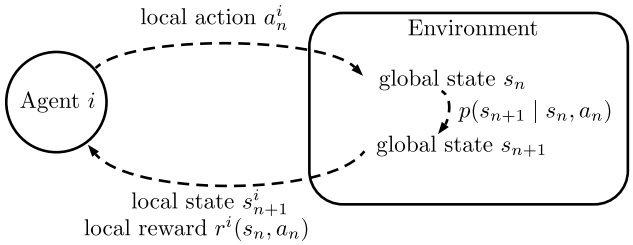


FIGURE 4. Markov game feedback loop over discrete time *n*.

the environment typically transitions to a new state as a function of all local actions and all local states. There are several additional properties to classify MARL settings, such as whether the setting is decentralized, or whether or to what extent agents transition independently [60]. We will discuss the two most common deep MARL algorithms in Section III-C. For a survey of MARL algorithm and various applications and challenges in computer networks, see [61] and [62].

5) RL WITH CONSTRAINTS

Contrained RL (CRL) [63] is a paradigm for constraint MDPs. The goal is to ensure that the agent's actions do not violate any environmental constraints. A set of constraints can be specified as hard (absolute and must always be satisfied) or soft (desired but can be violated if necessary) constraints. Safe RL (SRL) [64], on the other hand, aims to learn policies that minimize the likelihood of unsafe actions while maximizing the long-term expected reward for safe actions. Accordingly, CRL and SRL both focus on ensuring that the agent's actions do not violate certain constraints, but the formulation of constraints takes slightly different approaches. Both approaches are used when resources (such as bandwidth, computation, and energy) are limited [65], [66], [67] or when some applications may impose additional constraints (such as throughput and latency) [68], [69].

III. DEEP LEARNING—THE COOL KID OF ML

Deep Learning is a subfield of ML that aims at facilitating the learning of complex data representations by learning hierarchies of simpler intermediate representations [5], [70]. The resulting "stacking" of model blocks (predominantly Neural Network (NN) layers) is what gives *Deep Learning* its name. While the term *Deep Learning* has been around for decades, it only started to gain widespread traction in 2012 with the widely visible success of AlexNet [71] winning a widely popular image classification challenge with deep Convolutional Neural Network (CNN) (see Section III-B). Since then, the rate of progress concerning deep NN architectures, paradigms and learning techniques has skyrocketed, and the development of specialized hardware such as high-end GPUs or TPUs has led to deep learning models with millions or even billions of parameters [72]. As a consequence, a growing proportion of ML applications nowadays employ deep NN architectures. For networking applications, NNs are a powerful tool to learn non-linear relationships for complex problems. We shall now begin with a review of the most important NN background.

A. NEURAL NETWORKS

Given its central role in almost any cognitive process, neuroscientists have long tried to understand the inner workings and mechanisms of the brain. In [73], a mathematical model for a neuron was introduced that has since inspired an emerging class of ML model architectures: Artificial NN [5]. In NNs, a neuron j receives inputs a_i from nodes $i = 1, \ldots, n$ and a bias input $a_0 = 1$, and first computes a weighted sum using link weights w_{ij}: $a'_j = \sum_{i=0}^{n} w_{ij} a_i$. Then, it computes the output (also called *activation*) $o_j = g(a'_j)$ using an activation function g [74]. If multiple such neurons (mostly called *perceptrons* in the ML community) are connected in a directed and acyclic manner, they form a so-called feed-forward network that is usually arranged in layers. In such layered feed-forward networks, each neuron receives the outputs of the neurons of the previous layer, with the first layer receiving the overall model input and the last layer providing the overall model output. The resulting compound function expressed with the network can be highly complex, in fact, it is shown in [75] that with as little as one intermediate neuron layer and by choosing any squashing activation function (e.g., a non-decreasing function converging towards 0 and 1 on its respective ends), NNs can theoretically approximate any continuous function uniformly on any compact set to an arbitrary degree of accuracy. This statement is also known as the *Universal Approximation Theorem (UAT)*.

The UAT becomes even more interesting once we view the entire NN as a function $h_w(x)$ of the input vector x and the NN weights w [5]. The UAT implies that there exists a weight parameter configuration that sufficiently approximates the function which describes the desired solution to a problem. Hence, many learning problems can be viewed as a problem of *function approximation* to find the right NN weights. The most common technique to update the NN weights is gradient descent in combination with the so-called *backpropagation* algorithm [76]. For example, to update a NN using an input-output tuple (x, y), backpropagation calculates the derivative $\frac{\partial}{\partial w}|y - h_w(x)|^2$ of the output error with respect to the NN weights. This calculation is done sequentially starting from the output layer by applying the chain rule on the above derivative. Calculated gradients are then used to update the NN weights iteratively. Various algorithms have been proposed throughout the last decade to use the aforementioned calculated gradients most effectively. The most well-known algorithm is the ADAM optimizer [77], which adaptively selects the stepsize for individual NN weights based on the calculated gradient information. See [78] and the reference therein for various

other gradient-based methods. Note that most tools (such as PyTorch and TensorFlow) offer these optimimzers as black boxes without having to deal with the implementation details.

B. DEEP NEURAL NETWORK ARCHITECTURES

The UAT seems to advocate that rather simple feed-forward NN architectures can be used for any problem that might be solvable with ML. In practice, however, the findings of the UAT are greatly humbled by the excessive amounts of training data, the size of the NN models, and the required time for training necessary to achieve satisfactory results on a complex task. Furthermore, for many tasks, it can be observed that the members of the underlying data domain are semantically composable into simpler entities, spanning a hierarchy of concepts. As a consequence, researchers have started to add more structure to their models.

Different model architectures have proven effective for different tasks. An overview of common deep learning model architectures is given in Table 4. In the following subsections, we briefly present the most popular model archetypes and refer to the provided references for further reading. Interestingly, all of the model archetypes introduced below are derivable from the same basic mathematical framework and only differ in the shape of data and the assumptions made about regularities in the data [79].

1) MULTILAYER PERCEPTRONS (MLP)

Standard Deep Neural Networkss (DNNs) consisting of multiple hidden layers of neurons are also called Multilayer Perceptrons (MLPs). Together with non-linear activation functions, they have long become a standard tool for processing vector-shaped inputs as their hierarchy of non-linear function approximators is widely applicable across many problem domains [80]. However, given that Multilayer Perceptronss (MLPs) make rather few assumptions about the input and output data despite being shaped as a vector, for many tasks these models often perform unfavorably compared to more specialized models of similar size. A detailed introduction to MLPs is given in [70]. MLPs have already been used in computer and wireless networks, e.g., for channel decoding [81], in resource allocation [82], and in intrusion detection [83].

2) CONVOLUTIONAL NEURAL NETWORK (CNN)

In many problem domains, data exists on a grid-like structure where spatial patterns carry the same semantic information regardless of their location in the grid (also referred to as *translation invariance*). Examples include images (2D grids) but also time-series data (1D grids). To exploit this symmetry, Convolutional Neural Network (CNN) utilize spatial convolution, which applies the same learnable spatial parametric kernels (i.e., small matrices with learnable individual entries) on evenly spaced patches of the input grid [70]. The re-usage of a set of such kernels across multiple image positions is called *weight sharing* and greatly reduces the number of parameters needed to learn and extract the patterns of the input data.

3) RECURRENT NEURAL NETWORK (RNN)

For dealing with sequential data such as time series, Recurrent Neural Network (RNN) elements such as the Long Short-Term Memory (LSTM) [84] or the Gated Recurrent Unit (GRU) [85] have proven very useful. The commonality between all RNNs is feeding a portion of the output back into the RNN block for subsequent computations, enabling NN architectures with recurrent elements to capture sequential dependencies within the data [70].

4) GRAPH NEURAL NETWORK (GNN)

Recently, Graph Neural Networks (GNNs) have emerged as powerful architectures for handling graph-structured data. Utilizing permutation-invariant aggregation/pooling operations and permutation-equivariant message passing operations to learn patterns in the data while respecting the graph topology rather than assuming any specific ordering of its nodes and edges [86].

5) GENERATIVE ADVERSARIAL NETWORK (GAN)

Generative Adversarial Networks (GANs) [87] have emerged as a powerful tool for generating realistic data samples, including images [88], videos [89], and audio [90], but also network traffic [91]. GANs consist of two NNs: a generator that creates synthetic samples, and a discriminator that tries to distinguish between real and fake samples. These two networks are trained simultaneously in an adversarial setting, where the generator tries to fool the discriminator, while the discriminator tries to correctly identify the real samples. In the context of computer networks, GANs are useful for generating synthetic network traffic patterns that mimic real-world traffic. This is useful for testing and evaluating network performance metrics, intrusion detection systems, and network security protocols. See Section IV-B3 for example applications of GANs being used to generate data.

6) TRANSFORMERS

In many ML domains with complex long-range dependencies within data points, the *attention* mechanism [92] and its implementation in the Transformer architecture [93] has proven to be extremely powerful. Works like [94] and [95] show that Transformers can outperform both CNNs and RNNs in problems with spatial and temporal data as each token/component of the input can relate to any other component. While we refer to [93] for a detailed explanation of the attention mechanism, it is worth noting that transformers are a special case of GNNs operating on a fully-connected computation graph [96]. This implies that for large inputs, using the transformers is computive intensive.

7) LARGE LANGUAGE MODEL (LLM)

Large Language Models (LLMs) have recently gained significant attention in the field of Natural Language Processing (NLP). These models are trained on vast amounts

TABLE 4. Overview on deep learning architectures.

Data	Architecture	Task	Use Case Examples
Tabular (2D)	Multilayer Perceptrons (MLP)	Classification/Regression	Anomaly Detection
Time Series (3D)	1D Convolutional Neural Network (CNN)	Classification/Regression	Estimate network congestion
	Recurrent Neural Network (RNN)	Generative/Forecasting	Forecast network traffic
	Transformers	Representation Learning	Unsupervised representation learning of network traffic
Graph	Graph Neural Network (GNN)	Graph Classification Node Classification Edge Classification	Overall network performance Congestion at node Packet delay on network link
Any	Generative Adversarial Network (GAN)	Generative Modeling	Packet stream generation

of text data and can generate human-like text based on their input. One popular type of LLMs is the generative pre-trained transformer (GPT), which uses a transformer-based architecture and is pre-trained on large amounts of text data using a self-supervised learning approach. During pre-training, the model learns to predict the next word in a sentence, which enables it to generate coherent and contextually relevant text. GPTs can be fine-tuned on specific NLP tasks, such as text classification, summarization, and translation, by adding a task-specific output layer and training on a smaller dataset. Unlike traditional NLP models that rely on hand-crafted features, LLMs learn to represent the meaning of words and phrases in a continuous vector space, enabling them to perform a wide range of NLP tasks. In the context of computer networks, LLMs and GPTs have been used, for example, to generate synthetic network traffic [97], to explain decisions in intrusion and anomaly detection systems [98], [99] and for managing networks [100]. For an overview of applications, techniques, and challenges, we refer to [101].

8) GENERATIVE AI (GENAI)

Generative AI (GenAI) is a broader concept that can apply to any type of data [102]. It uses ML models, such as GPT, GAN, or/and others, to learn the patterns and structure of the given training data, and can then be used to generate realistic and novel outputs that are similar but not identical to the data. Additionally, and closely related, GenAI can be used with retrieval-augemented generation (RAG) [103] to automate collecting information from the network, analyze it, and push new configuration if necessary [104], [105], [106]. This removes the pain of learning a new documentation or writing new scripts, and simplifies the user interaction. Recent GenAI models have shown impressive and/or human-like capabilities in an unprecedented range of downstream tasks. As a consequence, several network industrial companies have started to develop or adjust commercial products that leverage GenAI e.g. to generate threat intelligence reports [107], security policies, incident response plans [104], and proactively identify and fix network issues [105].

C. DEEP REINFORCEMENT LEARNING (DRL)

Deep Reinforcement Learning (DRL) refers to the use of DNNs as function approximators for RL algorithms. The general idea of RL with function approximation has been briefly described in Section II-D2. With the advent of deep learning libraries such as Keras and TensorFlow (see Section IV), as well as standardized APIs such as Gymnasium (formerly known as OpenAI Gym), training of DRL algorithms has become very accessible. However, the success of RL with DNNs[4] relies on some key techniques. This subsection focuses on the most important DRL algorithms and the tools and techniques to train DRL models.

First, we will explain two key techniques for DRL based on Deep Q-Learning also known as Deep Q-Networks (DQN) [109]. DQN is a DRL algorithm for MDPs with finite action space. DQN seeks to approximate the optimal Q-function by a DNN $Q^\theta(s, a)$ with parameters θ. Specifically, a DQN takes a state s as input and outputs $Q^\theta(s, a)$ for every action a of the finite number of actions. The key techniques introduced for DQN are an *experience replay buffer* and a so-called *target network*. During training, DQN interacts with its environment, generating data tuples (s, a, r, s'). These data tuples are stored in an experience replay buffer. During training, DQN samples a mini-batch from this memory and applies a stochastic gradient descent step of the average squared Bellman error of the samples from the mini-batch. This rather simple technique reduces the bias of Q-Learning towards its recent interaction with the environment and thereby helps to stabilize training. In NN terminology, the right-hand side of the Bellman loss, i.e., $r + \gamma \max_{a'} Q^\theta(s', a')$ is the training target for $Q^\theta(s, a)$ given the data tuple (s, a, r, s'). In other words, the DQN itself is used to compute its training targets. The idea behind target networks is to use a separate target network $Q^{\theta'}(s, a)$ to compute the aforementioned training targets. The target parameters θ' are then chosen to track the actual training parameters slowly. With this, target networks provide more

[4]Historically, it was known that training RL with a NN could potentially lead to systematic overestimation of utility values (such as the Q-function) and thus to failed learning [108].

stable training targets, which has been shown to generally improve DRL training, see [109] and [110]. However, more recent theoretical and numerical studies suggest that gradient clipping is superior to the use of target networks [111].

DQN is also an integral component of the Deep Deterministic Policy Gradient (DDPG) algorithm [110], which is one of the most well-known actor-critic algorithms for continuous action spaces. In DDPG, a critic is trained using the DQN algorithm, while a deterministic policy is trained to maximize the approximated Q-function. DQN and DDPG, in turn, are the basis for the two common deep MARL algorithms Independent Deep Q-Learning [112] and Multi-Agent DDPG (MADDPG) [113]. However, only DDPG has a truly distributed version that can be run with nearly arbitrary communication delays over a communication network. This is known as the Distributed DDPG (3DPG) algorithm [114].

Another important technique for successful DRL training was proposed as part of the deep actor-critic algorithm Asynchronous-Advantage-Actor-Critic (A3C) [52]. The asynchronous part refers to using several agents in parallel simulated environments to improve and speed up DRL training. In other words, the training progress of several agents on the same problem is combined to enhance the training performance. This is especially important for complex tasks since multiple parallel processors can significantly reduce the overall training time.

The success of DRL has been demonstrated across various sub-areas in computer networks like management of satellite-terrestrial networks [115], multi-objective service coordination [116], scheduling for large-scale networked control systems [117], acoustic sensor networks, [118], adaptability of wireless sensor networks [119] and other applications in communications and networking [120].

1) GENERAL ADVICE FOR TRAINING DRL AGENTS

Training a DRL Agent to successfully solve a given problem can be a challenging task. In this section, we provide some general advice from our experience in the hope to ease this task.

1) It is good practice to normalize the states and actions, e.g., $[-1, 1]^d$, $d \in \mathbb{N}$. Linear scaling always makes this possible when the state space $\mathcal{S}$ is bounded in real dimensional space. When $\mathcal{S}$ is unbounded, let's say $\mathbb{R}^d$, one needs to use, e.g., a scaled version of the hyperbolic tangent or the inverse stereographic projection. Such nonlinear transformations, however, change the environment, and the resulting policies may perform poorly in the actual environment if the normalization is not chosen carefully. Ideally, one should aim at linear scaling throughout the state space's "expected" dominant part. As the action space is typically bounded, action normalization is less problematic.
2) Reward normalization should be used even more carefully than state normalization. In general, changing the reward changes the perception of an agent about the environment and results in different learned policies.
3) *The design of the reward signal* is an integral part of the design of an MDP. One has to craft a reward function that incentivizes the desired behavior to get an algorithm to learn the desired goal. Some additional comments in no particular order: Make it easy for an agent to distinguish good from bad scenarios; Continuous rewards or dense rewards typically make it easier for algorithms to learn; If possible, avoid sparse rewards and instead shape the rewards to give gradual feedback; Strictly positive rewards incentive agents to avoid terminal states; Strictly negative rewards incentive agents to reach terminal states.
4) It should be avoided to train DRL models with drop-outs. Drop-outs is a regularization technique that was introduced in [121] to train NN models with less overfitting while improving the generalization. However, this leads to increased training variance, which is generally undesirable for the training of DRL.

2) ALGORITHM CATEGORIZATION

The sheer amount of available DRL algorithms can be overwhelming for starters in the field, making it challenging to find appropriate algorithms for a given problem. To ease the algorithm selection, we provide categorizations of widely used single-agent and multi-agent DRL algorithms in Figure 5 and Figure 6, respectively. We note that the tree structures are simplified. For example, the model-based algorithms in Figure 5 can further be classified into value-based, policy-based, actor-critic and on/off-policy algorithms. Furthermore, only selected and widely used algorithms are shown. These categorizations should serve as starting points. The final algorithm selection for a specific problem should also consider additional factors such as sampling efficiency, algorithm stability and exploration strategy.

Single-Agent-DRL Algorithm Categorization: Single-agent-DRL algorithms can be coarsely categorized by their supported action space (discrete/continuous), if they are model-based or model-free, and if they are value-based, policy-based or a combination of both - called actor-critic. Considering the tree-structure in Figure 5, it can be seen that some algorithms (e.g., A2C/A3C, SAC, PPO) can be used for both, discrete and continuous action spaces, while others, such as DQN and DDPG, are only compatible with one of them.

MARL algorithms can generally be categorized based on the same factors as single-agent DRL algorithms. However, additional multi-agent based factors can be included. These are mainly centralized/decentralized learning and cooperative/independent learning. To preserve clarity, some of the traditional single-agent based factors have been omitted in Figure 6.

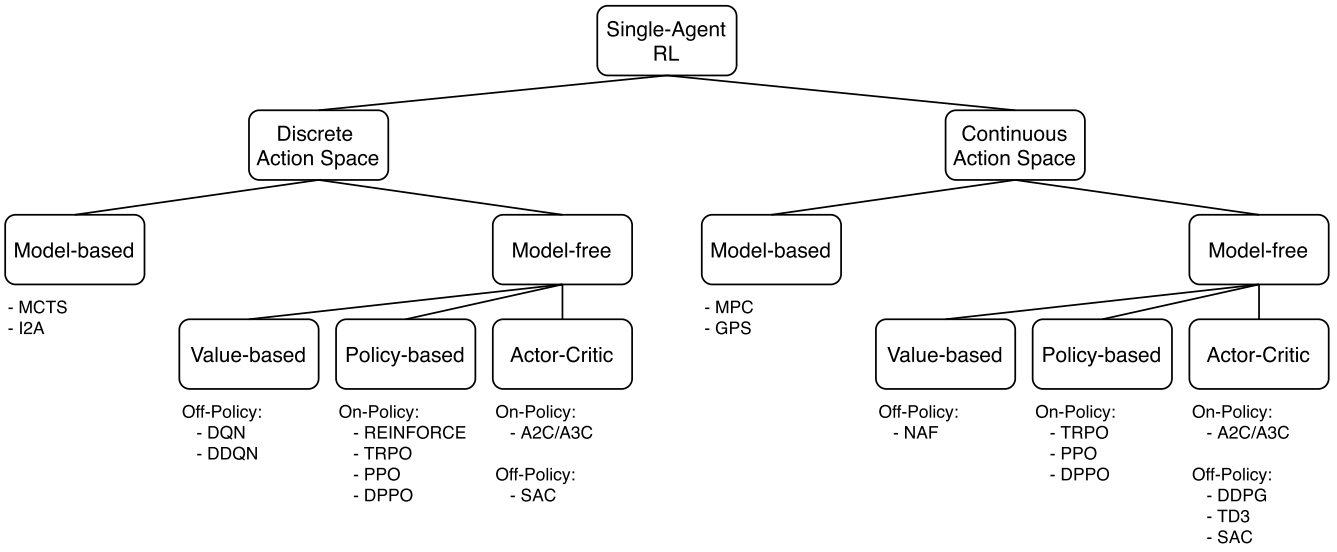


FIGURE 5. Categorization of well-used Single-Agent Deep Reinforcement Learning (DRL) algorithms.

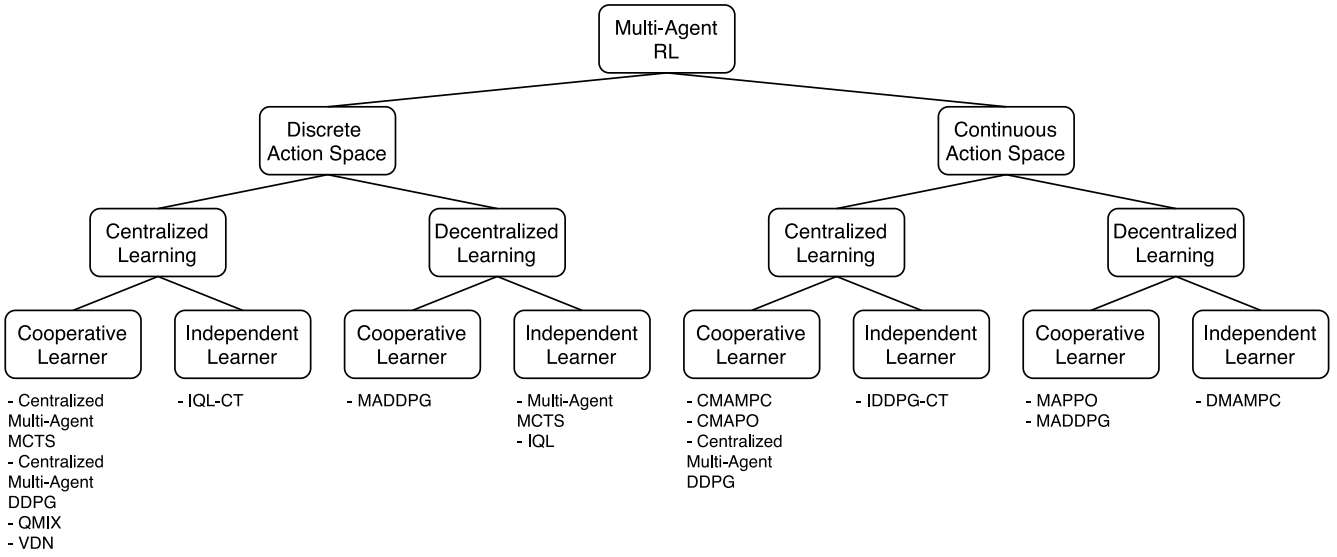


FIGURE 6. Categorization of well-used Multi-Agent RL (MARL) algorithms.

IV. DATASETS, TOOLS, AND FRAMEWORKS

Now that we have discussed what ML is and its potential applications, we will introduce here the most popular datasets in the field of networks, as well as emulators, and simulators that can be used to run ML experiments. Since ML models parameters are learned from data, the datasets used are crucial in accomplishing the intended task, such as network latency prediction or decision-making for traffic routes.

Additionally, ML models need to be tested before being applied in a productive environment. Thus, well-known network tools and frameworks can aid in prototyping, tracking, and evaluating these models.

A. DATASETS

Datasets are usually not plug-and-play and require preprocessing. The type of preprocessing required for the datasets depends on the specific problem being addressed and the type of data being used. In general, preprocessing includes the following steps:

- Data cleaning: This involves removing any missing, inconsistent, or irrelevant data to ensure the quality of the data being used for training.
- Data normalization: This involves transforming the data into a common scale, such as normalizing the values between 0 and 1, to ensure that no variable has an undue influence on the model.
- Data selection: This involves selecting the relevant features or variables from the dataset that are most important for the problem at hand. This step is important to reduce the dimensionality of the data; making it easier to improve the performance of the model; it removes irrelevant or redundant features, it can help to speed up the training process and reduce the computational resources required for analysis.

- Data transformation: This involves transforming the data into a format suitable for the ML algorithm being used, such as converting categorical variables into numerical values using one-hot encoding; it generates a vector, whose length corresponds to the number of categories in the dataset. Data points belonging to the category are assigned 1, otherwise 0.
- Data splitting: This involves splitting the dataset into a training set to train the model, a validation set to evaluate its performance during training, and a test set to evaluate the models' performance after training.

It is important to note that the specific preprocessing steps required may vary depending on the dataset, the problem being addressed, and the type of ML model used. The preprocessing steps should be chosen carefully to ensure that the data is suitable for training and that the model can accurately represent the underlying relationships in the data. In the following, we present the most popular network domain datasets in the literature for different applications.

1) MOBILE NETWORK THROUGHPUT DATASETS

A common problem in networking research is replicating realistic network conditions, especially throughputs. Dynamic Adaptive Streaming over HTTP (DASH) is one such exemplary research area. Depending on the mobile network, different datasets containing traces of real-world measurements have been created in order to allow for a better comparison between different research approaches.

For 3G mobile networks, the dataset by Riiser et al. [122] is widely used [123]. It contains 86 traces from measurements conducted on commute paths in Oslo, Norway, using six different mobility patterns (cf. Table 5). Besides the download throughput, it also contains the GPS latitude and longitude coordinates of the measurement device.

For 4G mobile networks, the dataset by Van Der Hooft et al. [124] (we call it *4G_a* in this paper) is commonly used [125]. It contains traces of 40 measurements with different mobility patterns (cf. Table 5) conducted in Ghent, Belgium. It is similar to the 3G dataset by containing the download throughput, as well as the GPS coordinates of the measurement device.

Another widely used [126] dataset for 4G networks was created by Raca et al. [127] (we call it *4G_b* in this paper). A total of 135 measurements were conducted in Ireland. In comparison to the 4G_a dataset, this one is larger, also contains different mobility patterns (cf. Table 5), and contains significantly more metrics, such as the download and upload throughput, additional channel-related metrics, context-related metrics, and cell-related metrics.

In Farthofer et al. [128] an LTE dataset for the use of ML is described. The dataset is measured on an Austrian highway and contains over 2000 measurement points per month over a time period of two years. Additionally, there are different signal parameters measured in the dataset like SINR, RSSI, and RSRP as well as GPS data, time, data rate, etc., and it is published at CRAWDAD.[5]

Raca et al. also created a widely used [129] dataset for 5G networks [130]. It contains 83 traces of measurements in Ireland with two different mobility patterns (cf. Table 5). The measurement setup and dataset structure are comparable to their 4G_b dataset. In addition, it also contains ping statistics.

Table 5 provides a comparative overview of the presented datasets.

2) ROUTING

As routing is an important part of networking, having a real-world dataset for it can be beneficial for training ML models, evaluating their performance and testing their robustness in the case of network failures and other real-world issues. In the following, we discuss some of these datasets.

The Abilene dataset [131] is a real-world network trace that captures the communication patterns of a backbone network. It was collected by the Abilene project, which was a collaboration between researchers from the University of California, San Diego, and the University of Kansas. The Abilene dataset provides information about the communication patterns of a backbone network that connects several research institutions in the United States. It includes information about the network topologies, routing algorithms, and traffic patterns of the network. It contains information about the routes taken by packets, the number of packets sent and received, and the size of the packets. The dataset is commonly used for research in the areas of network routing, network management, and network performance evaluation.

The Global Environment for Network Innovations (GENI) dataset [132] provides network traces from real-world deployments. GENI is a large-scale research infrastructure that provides a platform for conducting experiments and evaluating new network technologies and protocols. The dataset includes network traces that were collected from a variety of testbeds and networks, including campus networks, data centers, and wide-area networks. It includes information about the network topology, routing algorithms, and traffic patterns of the network. It also contains information about the routes taken by packets, the number of packets sent and received, and the size of the packets. The dataset is commonly used for research in the areas of network routing, network management, and network performance evaluation.

The Cooperative Association for Internet Data Analysis (CAIDA) anonymized internet traces dataset is a collection of network traces that were collected by the CAIDA project [133]. CAIDA is a non-profit research organization that collects and analyzes data about the internet to gain insights into its structure and behavior. It includes data from a variety of sources, including routers, switches, and end hosts. The data includes information about the network topology, routing algorithms, and network traffic patterns. Additionally, it contains information about the routes taken by packets,

[5] https://www.crawdad.org/

TABLE 5. Overview of throughput datasets for 3G, 4G, and 5G mobile networks.

	3G (Year 2013) [122]	4G_a (Year 2016) [124]	4G_b (Year 2018) [127]	5G (Year 2020) [130]
Measurement method	HTTP video streaming	HTTP file download	TCP file download, Youtube streaming	TCP file download Netflix video streaming Amazon video streaming
Number of traces	86	40	135	83
Trace lengths (min;max)	195 s; 12224 s	166 s; 758 s	383 s; 11141 s	266 s; 7752 s
DL throughput (min; max)	0; approx. 9 Mb/s	0; approx. 111 Mb/s	0; approx. 173 Mb/s	0; approx. 533 Mb/s
UL throughput (min; max)	/	/	0; approx. 4 Mb/s	0; approx. 7 Mb/s
Measurement interval	1 s	1 s	1 s	1 s
Mobility patterns	Metro, Tram, Train, Bus, Ferry, Car	Foot, Bicycle, Bus, Tram, Train, Car	Static, Pedestrian, Car, Bus, Train	Static, Car
Logged metrics	• GPS latitude, longitude • DL throughput	• GPS latitude, longitude • DL throughput	19 metrics, e.g.: • GPS longitude, latitude • DL & UL throughput • RSSI, RSRP, RSRQ • CQI, SNR	25 metrics, e.g.: • GPS longitude, latitude • DL & UL throughput • RSSI, RSRP, RSRQ • CQI, SNR

the number of packets sent and received, and the size of the packets. The data is collected using active and passive measurements:

Active measurement involves actively sending test packets or requests to a network, and then analyzing the resulting responses to gain insight into the network behavior and performance. Examples of active measurement techniques include pinging, tracerouting, and bandwidth testing. Active measurements are often more accurate and provide more detailed information about the network, but they can also introduce more overhead into the network and disrupt normal network traffic.

Passive measurement involves observing network traffic without actively generating any test traffic. Passive measurements are typically less disruptive to the network and do not introduce any overhead, but they provide a more limited view of the network's behavior and performance. Examples of passive measurement techniques include network traffic analysis, packet capture and analysis, and log file analysis.

The RocketFuel dataset [134] is a collection of network topology data that was collected by the RocketFuel project. The RocketFuel project was a research effort aimed at studying the structure and behavior of the Internet at the level of individual routers and links. The project collected data from several large Internet Service Providers (ISPs) and used it to create a high-resolution map of the Internet. It includes information about the network topology and the paths that packets take through the network. It also includes information about the capacities of the links between routers, as well as information about the location and characteristics of the routers themselves.

The Internet Topology Zoo [135] is a collection of network topology datasets that provide information about the physical structure of different networks. The datasets in the Internet Topology Zoo come from a variety of sources, including measurements of the internet, testbeds, and simulations. Its datasets provide information about the connections between nodes in a network, the capacities of the links between nodes, and the characteristics of the nodes themselves, such as their locations and capabilities. One of the main strengths of the Internet Topology Zoo is its comprehensive coverage of different types of networks, including wide-area networks, data centers, and other large-scale networks. This makes it a valuable resource for researchers and practitioners working in the field of network routing and network management, as it provides a diverse set of datasets for evaluating and comparing different algorithms and technologies.

To sum up, GENI and Abilene datasets primarily focus on network infrastructure, providing researchers access to national research networks. Conversely, CAIDA and RocketFuel are designed to facilitate the measurement and analysis of network traffic and topology. The Internet Topology Zoo, meanwhile, is a collection of publicly available network topologies that researchers can use for various purposes. Thus, the size of the network varies depending on the scope and focus of the dataset. The GENI and Abilene datasets tend to cover larger networks compared to CAIDA and RocketFuel, which prioritize measurement and analysis tools [136]. CAIDA and RocketFuel datasets use passive measurements, while GENI and Abilene datasets use both active and passive measurements. The Internet Topology Zoo is a collection of network topologies and does not involve any measurements. Further comparison is shown in Table 6

3) DYNAMIC ADAPTIVE STREAMING OVER HTTP (DASH)

Video streaming via Dynamic Adaptive Streaming over HTTP (DASH) is a large research area in networking. Recent rate adaptation algorithms often aim to optimize the user's Quality of Experience (QoE) under the given network conditions, such as a constrained bandwidth [137]. While these algorithms were initially conventional heuristics, DRL-based approaches have recently shown excellent performance and are now considered state-of-the-art [138]. In order to benchmark different solutions, publicly available DASH datasets are often used [138]. Typically, the datasets contain

TABLE 6. Overview of available routing datasets.

Point of comparison	Abilene [131]	Global Environment for Network Innovations (GENI) [132]	Cooperative Association for Internet Data Analysis (CAIDA) [133]	RocketFuel [134]	Zoo [135]
Scope	Abilene network; a major US backbone network in the early 2000s, used for research and education network–Last updated in Nov 2022 [131]	various types of networks, including the wide-area, data centers, and campus networks. The dataset is updated on regular basis.	Provides data specifically about the Internet, collected through a combination of active and passive measurement techniques. The age of these datasets varies, with some datasets being ongoing and others being completed or one-time snapshots	Provides data about the ISP topologies and other types of networks. It was last updated in 2004 for measuring ISP topologies [134]	Provides data about a variety of different types of networks, including data centers and other large-scale networks, obtained from a variety of sources. It is being updated on regular basis with the most recent update in April 2021.
Data Collection	network measurements and simulations.	measurements, simulations, and testbeds.	active and passive measurement techniques.	measurements and simulations.	measurements, simulations, and testbeds.
Data Availability	Available for research and educational purposes.	Available for research and educational purposes.	Available for research purposes, but access may be restricted for some datasets.	Available for research and educational purposes.	Available for research and educational purposes.
Data Format	standard graph format.	raw data and processed data.	raw data and processed data.	standard graph format.	standard graph format
Data Accuracy	may vary depending on the source of the data and the methods used to obtain it.	may vary depending on the source of the data and the methods used to obtain it.	Strive to provide accurate and up-to-date data.	may vary depending on the source of the data and the methods used to obtain it.	may vary depending on the source of the data and the methods used to obtain it.
Scale	relatively small-scale network	data about a large-scale, diverse testbed	very large and complex networks	a limited number of individual networks	mix of small- and large-scale networks

videos that are encoded under a controlled set of parameters, e.g., resolution and bitrate, and split into segments of certain lengths. The solutions are commonly evaluated via simulations where the videos from the DASH datasets are streamed over simulated networks [139]. Realistic network conditions, especially the download bandwidth, are commonly simulated using the network traces from the datasets presented in Section IV-A1 [138]. In the following, we present four commonly used DASH video datasets. Table 7 provides an overview of their most important properties.

The DASH dataset [140] is an old (2012) but still widely used dataset, e.g., to test new QoE-schemes [139]. It contains 6 videos of different genres split into segments ranging from 1 - 15 seconds in length.

The Distributed DASH (D-DASH) dataset [141] was published in 2013 and is intended to be used in real-world testbeds. It contains one video that is distributed on servers in Klagenfurt, Paris, and Prague. This enables a client to choose the requested location for each segment individually.

The ultra high definition HEVC DASH dataset [142] was published in 2014 and includes one video. In contrast to the DASH and D-DASH datasets, the video is encoded with the newer and more efficient H.265 (HEVC) video codec. Furthermore, it is encoded in UHD resolution, at 30 and 60 Frames per Second (FPS), and at 8 and 10 bits.

The multi-codec DASH dataset [143] is a rather new dataset from 2018. It consists of 10 videos that are encoded with four different video codecs: H.264, H.265, VP9, and AV1. In addition, three different video FPS are included: 24, 30, and 60.

4) MOBILITY AND AUTONOMOUS VEHICLES

In the context of mobility or autonomous driving using a wireless network infrastructure (let it be cellular or V2X), most of the studies in the literature discussing solutions and their results do not make the datasets publicly available for scrutiny by third parties. As such, the results are difficult to verify and validate properly. Nevertheless, many studies rely on simulated datasets. An open-source traffic simulation software called Simulation of Urban Mobility (SUMO)[6] provides datasets for simulating realistic urban traffic scenarios. Among the datasets are road networks, traffic demand patterns, and vehicle behavior models, which can be customized for different traffic scenarios and urban environments [144].

Although SUMO is popular among researchers and practitioners in the industry, another software called Multi-Agent Transport Simulation (MATSim)[7] is often used in academic research. While SUMO focuses on macroscopic traffic flow modeling, MATSim uses an agent-based approach to model individual travel behavior [145]. As a result, MATSim can capture more complex individual decision processes, while SUMO is better suited for overall traffic flow modeling. Another open-source software is CityFlow,[8] which includes a range of features that are not available in SUMO, such as real-time simulation and the ability to model pedestrian and bicycle traffic [146].

There exist other alternatives, yet commercial, that also provide mobility datasets, such as Aimsun,[9] Vissim[10] and TransModeler[11] We present in Table 8 an overview of these datasets, while a comparison of the use cases for some of these datasets was shown in [147].

[6]https://sumo.dlr.de/docs/Data/Scenarios.html
[7]https://www.matsim.org/open-scenario-data
[8]https://github.com/cybercore-co-ltd/track2_aicity_2021
[9]https://www.aimsun.com/
[10]https://www.ptvgroup.com/
[11]https://www.caliper.com/transmodeler/default.htm

TABLE 7. Overview of the DASH datasets.

	DASH (Year 2012) [140]	D-DASH (Year 2013) [141]	HEVC-DASH (Year 2014) [142]	Multi-Codec DASH (2018) [143]
Number of Videos	6	1	1	10
Video Content	Animation, Sport, Movie	Sports	Movie	Animation, Moving head, Moving cars, Nature, Moving jockey, Moving horses, Composite, Rotating wind vanes, Moving yacht
Video lengths	586 s - 5848 s	5848 s	142 s	20 s - 888 s
Resolutions	320x240 - 1920x1080	320x240 - 1920x1080	1280x720 - 3840x2160	256x144 - 3840x2160
Bitrates	50 kbit/s - 8 Mbit/s	100 kbit/s - 6 Mbit/s (Video), 64 kbit/s - 165 kbit/s (Audio)	1.8 Mbit/s - 18 Mbit/s (Video), 128 kbit/s (Audio)	100 kbit/s - 20 Mbit/s
Segment Lengths	1, 2, 4, 6, 10, 15 s	2, 4, 6, 8, 10, 15 s	2, 4, 6, 10, 20 s	2, 4 s
Encoder	H.264	H.264	H.265 (Video), HEAAC (Audio)	H.264, H.265, VP9, AV1

TABLE 8. Overview of mobility frameworks.

Points of comparison	SUMO [148]	MATSim [149]	Aimsun [150]	PTV Vissim [151]	TransModeler [152]	CityFlow [153]
Focus	urban	urban with large-scale transport systems	urban and non-urban	small-scale urban	small towns and large cities	urban
Modeling	microscopic individual travel behavior	macroscopic traffic flow modeling	microscopic	microscopic	macroscopic and microscopic	macroscopic and microscopic
Output	inter-modal information on traffic flow and congestion	data on individual vehicles	data on individual vehicles	data on individual vehicles	data on individual vehicles, GIS mapping and 3D visualization	individual vehicle behavior
Availabilty	open-source	open-source	commercial	commercial	commercial	open-source

5) (ENCRYPTED) NETWORK TRAFFIC ANALYTICS

Another common task for ML in networking is network traffic analytics. This includes the task of traffic/service classification, i.e., identifying an active service or traffic type in the network. Examples for such a task are distinguishing between video and web traffic, between services like YouTube and Netflix, or even between different Android apps. Due to pervasive encryption with for example TLS on application and transport layer, protocols like HTTPS, DNS over TLS (DoT), and QUIC [154] do not yield sufficient unencrypted data that reliably identify services and traffic types. Instead, new techniques have to be developed which make use of the available unencrypted data. For encrypted network traffic analytics, a common approach is therefore to extract packet sizes, directions, and inter-arrival times as well as potential additional information like port numbers to build features. These features describe the network traffic of a specific service or traffic type [155], [156], [157], [158]. These features are then fed to ML models to learn specific patterns exhibited by different traffic types or services. Beyond traffic classification, this type of analytics is often used for security-related tasks like intrusion detection or fingerprinting of websites, browsers, devices, and operating systems, and for estimating the QoE of services [159], [160]. Due to the prevalence of those topics, there also exists a variety of datasets for the different network traffic analytic tasks.

An overview on the topic of traffic classification along with a list of existing works (and solutions) and datasets is provided in [161]. However, many of these datasets are quite old and, thus, outdated. A dataset for encrypted network traffic classification of YouTube and Netflix is provided in [162]. Here, the authors collected three classes of flows, namely, web flows, YouTube flows, and Netflix flows for the most popular websites and videos, while using different end devices, browsers, and operating systems. A new dataset for app traffic classification is Mirage [163]. This dataset was generated using three different mobile devices, which were used by real experimenters (students) once or twice a day. Overall, each experimenter generated 12 captures of a duration of 5 to 10 minutes. Experimenters were hereby instructed to use the app as they would usually do in their day-to-day life. The resulting datasets consists of 40 Android apps from 16 different categories. Another new dataset for traffic classification of mobile apps is AppClassNet [164]. It was designed as ImageNet for encrypted network traffic analytics and, therefore, is significantly larger in terms of tested apps and available samples than other datasets.

The corresponding public dataset contains 500 apps with a volume of around 10 TB and stems from passive measurements.

For security tasks, in particular network intrusion detection, a comprehensive survey can be found in [165]. In this survey, the authors describe over 30 datasets and list the corresponding attacks and data types. A variety of datasets for different security tasks is also provided by the Canadian Institute for Cybersecurity, University of New Brunswick (UNB) [166]. They provide more than 25 datasets from different categories. These categories include IoT, dark web, DNS, IDS, traffic classification (web and apps using Tor or VPN), malware, and operational technology. A dataset for fingerprinting of devices and operating system in the potential presence of VPN is provided in [167]. The dataset contains around 20000 examples suitable for fingerprinting browsers, operating systems, and apps.

B. DATA GENERATION

Using real-world data is often challenging due to its limited availability and applicability. In this section, we explore the use of simulators, emulators, and synthetic methods, such as GANs, for generating data that can be used to train ML models. These approaches have the potential to help when datasets are not available or suited, and can enable the creation of diverse and complex datasets.

1) SIMULATION TOOLS

One of the paramount parts of designing a new scheme or protocol is the evaluation process. There are various methods, including real-world experiments, simulation, emulation, or analytical models in order to perform detailed investigation of the newly designed scheme. Nevertheless, each method has its advantages and disadvantages. When employing practical tests, the accuracy of the results can be faultless, however, in contrast, complexity and cost can increase. On the other hand, modeling a new protocol based on conceptualization is beneficial for having an analytics model, yet the complexity is still a flaw. Moreover, the accuracy can be declined as the lack of capability for reflecting real-world scenarios might be highlighted. Considering the above challenges, using simulation and emulation environments can strike a good balance between complexity, cost, and accuracy. They might not depict real-world conditions minutely, but even so, they are eminent tools that can assist a researcher or developer to expand novel schemes. A thorough analysis of different simulators for networking can be found in [168] and [169].

NETWORK SIMULATOR 3

One of the most powerful simulation tools in networking is ns-3 (Network Simulator - 3)[12] [170], which is a discrete-event open-source simulator under the GNU GPLv2 license. This tool comes with various modules such as Wi-Fi, LTE, or even a recently released mmWave (millimeter wave) [171] module to ease the way for researchers to have somehow reliable test environments for the newly developed approaches and reduce development time for various kinds of research interests. Indeed, ns-3 can assist researchers in network performance evaluation, however, if it is extended through open-source AI frameworks, the procedure would be more beneficial to ML problems as by default it does not support ML approaches. An attempt to do so was employed in ns3-gym [172], an extension of ns-3 connecting the module to the OpenAI Gym toolkit. This connection is done utilizing Zero MQ sockets through the IPC (Inter-Process Communications) method. Moreover, the capability and adaptability of OpenAI Gym to reinforcement learning can be favorable, as it is a widespread library such as TensorFlow and Scikit-Learn. ns3-gym aims at ameliorating the process of network prototyping that employs reinforcement learning. This module enhances the feature of scalability, which is important for having several instances in ns-3 and making the conversion and deployment of ns-3 scripts feasible in the OpenAI Gym. Furthermore, debugging and exploitation of the module could be kept at a level that is uncomplicated for users as it is such a conventional module for ns-3, having two main blocks of OpenAI Gym and ns-3 that interact with each other. Another interface extension that bridges the ns-3 and python-side ML implementation is ns3-ai [173], which claims to greatly increase the interaction speed by facilitating communication through a shared memory block.

OMNET++

Another popular discrete-event network simulator is OMNeT++,[13] which can be used free of charge for academic and educational purposes under a license with rights similar to the GPL,[14] but requires a paid license for commercial use. While OMNeT++ itself only contains the core simulation framework, various models can be added via external frameworks. The most important one is the INET Framework,[15] which is maintained by the OMNeT++ core team and provides models for network standards like IEEE 802.3 and IEEE 802.11 as well as higher layer protocols like IP, UDP, and TCP.

In terms of ML, Veins-Gym [174] exposes an OMNeT++ simulation as an OpenAI Gym environment, analogous to ns3-gym for ns-3. Despite its name, Veins-Gym can be used not only in combination with the Veins framework but also with any OMNeT++ simulation.

An overview with examples of how to use different ML frameworks such as TensorFlow in OMNeT++ can be found in [175].

2) EMULATORS

A network emulator, unlike a simulator, creates a virtual copy of a physical device, including all hardware and software

[12] https://www.nsnam.org

[13] https://omnetpp.org/

[14] https://omnetpp.org/intro/license

[15] https://inet.omnetpp.org/

TABLE 9. Overview of the encrypted network traffic datasets.

	YouTube/Netflix (Year 2021) [162]	Mirage (Year 2019) [163]	AppClassNet (Year 2022) [164]	Security (Year 2016) [167]
Number of Classes	Flow Types (3)	Apps (40)	Apps (500)	OS (3), Browser (5), Application (8)
Scope	Flow Classification	App Classification	App Classification	Fingerprinting
Data Availability	Available for research and educational purposes	Available for research and educational purposes	Available for commercial purposes	Available for research and educational purposes
Data Format	Raw data	Raw data	Preprocessed (Numpy)	Raw data

configurations, to functionally replace it. Hence, emulation is more accurate than simulation, but also more expensive in terms of computation resources. There are many network emulating tools, including but not limited to:

- Mininet[16]: a Python-based tool focused on emulating software-defined networks (SDNs) using OpenFlow switches.
- GNS3[17]: supports a wide range of network devices and protocols using virtual machines and real devices.
- Mahimahi[18]: a lightweight network emulator that is designed to emulate low-bandwidth networks with high latency.
- WANEM[19]: a Linux-based tool that can be used to emulate various network conditions such as latency, packet loss, and bandwidth limitations in WAN.
- TENS[20]: a VM tool that can be used to generate emulated network traffic for security evaluation purposes. It can generate various types of traffic, such as HTTP, FTP, SMTP, etc.
- CORE[21]: similar to GNS3 but with further emulation capabilities beyond traditional networks, such as SDN and virtualization technologies.
- FlowEmu [176][22]: a modular network link emulator with a flow-based programming inspired user interface that integrates TensorFlow for writing custom ML modules.

Many of these tools have been used for training and evaluating ML algorithms. For example, SDWAN-gym[23] and IROKO [177] are Python-based platforms built on top of Mininet for training and evaluating reinforcement learning algorithms in software-defined WANs and data centers, respectively. It is often the case that emulated data is mixed with real data for a large reliable dataset. There exist many datasets that adopt this approach in cybersecurity, such as the Canadian Institute for Cybersecurity database.[24] They provide the "CICIDS2017" dataset, labeled network flows with full packet payloads in PCAP format, for ML and deep learning purposes. Also, they provide the *AndMal 2020* dataset to identify and classify Android malware based on ML.

[16]http://mininet.org/
[17]https://docs.gns3.com/
[18]https://manpages.org/mahimahi/
[19]https://github.com/PJO2/wanem
[20]https://github.com/vmware/te-ns
[21]http://coreemu.github.io/core/
[22]https://github.com/ComNetsHH/FlowEmu
[23]https://github.com/amitnilams/sdwan-gym
[24]https://www.unb.ca/cic/datasets/index.html

3) SYNTHETIC

Synthetic data is needed because it can help to overcome the lack of up-to-date real-world data and privacy constraints, which limit the development of new models. In addition, synthetic data can provide an efficient mechanism to surmount the lack of labeled datasets and post-processing overhead. In the context of network traffic analysis, synthetic data can be used, for example, to train ML models to detect cyber-attacks and resolve network congestion as well as other performance issues.

SynGAN (Synthetic Generative Adversarial Network) [178] is a packet-level GAN designed to generate synthetic traffic data. It generates synthetic packets that closely resemble real-world traffic by simultaneously training the generator and discriminator networks. The generator network takes random noise as input and produces synthetic network traffic data as output, while the discriminator network distinguishes between synthetic and real data. Adversarial training ensures that the synthetic data produced by SynGAN is representative of real network traffic.

To make sure that the generated data satisfies certain constraints, PAC-GAN (Projection Adversarial Constraint GAN) [91] uses a projection operator to map the generated data onto a feasible set that satisfies the desired constraints. In addition to the standard GAN loss, PAC-GAN uses a constraint loss to ensure that the generated data is not only realistic but also satisfies the desired constraints.

Another type of traffic generator is flow-based GANs that, unlike packet generators, focus on individual packets and generate flows of packets that share common characteristics, such as source and destination IP addresses, source and destination ports, and protocol type. Additionally, they can reduce the amount of data needed to be generated by generating a single flow instead of multiple individual packets.

The authors in [179] propose different preprocessing approaches, for transforming IP addresses of flows into a continuous feature, since GANs can only process continuous features. Then, they use domain knowledge, such as packet size, inter-arrival time, and flow duration distributions, to evaluate the quality of the generated data. Another

example is MAIGAN (Massive Attack Generator via GAN) [180] that generates synthetic network traffic that mimics various types of cyber attacks, including Distributed Denial of Service (DDoS) attacks, port scanning attacks, and brute-force attacks, which is able to bypass black-box ML-based detection models.[25] To handle the problem of imbalanced traffic classification, i.e., data used for training the classification model contains a disproportionate number of samples from one class compared to the others, ITC-GAN [181] uses a modified GAN architecture with a class-balancing loss, based on the inter-packet time characteristics, which helps to balance the number of samples from each class during training and remove the bias in the classification model.

The work in [182] discusses other GANs models for network traffic generation, including Facebook Chat GAN [183] that generates chat message sequences based on Facebook Messenger data, ZipNet GAN [184] that generates compressed network packets using Huffman coding and PcapGAN [185] that generates network packet captures (pcap files) by learning from real-world pcap files.

C. MACHINE LEARNING TOOLS

Selecting the right tools to solve ML problems can be challenging. This section gives an overview of the most commonly used Python-based tools for classic ML, deep learning, and reinforcement learning. Furthermore, we provide guidelines for when to use which tool.

1) CLASSIC MACHINE LEARNING

Scikit-learn,[26] also known as *sklearn*, is a machine-learning library for Python that provides a wide range of tools for data analysis and modeling. It is built on top of other popular Python libraries such as NumPy[27] and SciPy[28] and is designed to be easy to use and efficient. Sklearn provides a variety of ML algorithms for various tasks such as classification, regression, clustering, and dimensionality reduction. These algorithms are carefully designed to be intuitive, fast, and efficient, allowing for quick prototyping and testing of ML models. Some of the commonly used algorithms available in sklearn include support vector machines, decision trees, k-nearest neighbors, logistic regression, and random forests. It also includes tools for model selection, preprocessing, and evaluation that enable researchers to preprocess data and select the right model for a problem. Some of the popular tools available in sklearn include cross-validation, grid search, PCA, and feature selection. A key advantage of sklearn is its wide range of algorithms for different tasks that are designed to be easily utilized by beginners to get started with ML.

2) DEEP LEARNING

Keras[29]: Keras is an open-source library for deep learning in Python that provides a high-level API for building, training, and evaluating DL models. A key advantage of Keras is its simplicity due to its high-level API, which abstracts away the complexities of building and training deep learning models. Keras provides a user-friendly API that makes it easy to create and experiment with different neural network architectures and offers a wide range of pre-trained models. Keras provides a wide range of pre-built layers, optimizers, and other building blocks that can be easily combined to create complex models. Keras can also be run on top of several backends, including TensorFlow, Theano, and CNTK. Keras is best suited for building simple to medium-complexity neural networks and deep learning models.

TensorFlow[30]: TensorFlow is an open-source library for ML and deep learning developed by Google. It is widely used for a variety of tasks, such as image and speech recognition, natural language processing, and for training and deploying large-scale ML models. The key advantages of TensorFlow are flexibility and scalability. It allows for the building and training of complex models, including deep neural networks, and can run on a variety of platforms, including CPUs, GPUs, and TPUs. TensorFlow can be used for distributed (ML) training across multiple machines, making it well-suited for large-scale ML tasks. The TensorFlow ecosystem includes a wide range of tools and libraries for tasks such as data pre-processing, model visualization, and deployment. TensorFlow is a good choice for building complex deep-learning models that require a high degree of customization and is suitable for both research and production environments.

PyTorch[31]: PyTorch is an open-source ML library for Python that is primarily developed by Facebook's AI research group. A distinctive feature of PyTorch is its dynamic computational graph, which allows for more flexibility in building and modifying models compared to the static computation graph used in other libraries, such as TensorFlow. This feature facilitates the easy implementation of advanced deep-learning models, including those with conditional logic, loops, and dynamic inputs. PyTorch is designed to be modular and flexible, with a wide range of building blocks (e.g., layers, activations, loss functions, and optimizers) that can be used to create custom deep-learning models with various complexity levels. Furthermore, PyTorch supports distributed training, allowing for the efficient use of multiple GPUs and machines. PyTorch is suitable for building complex deep learning models, and its dynamic computational graph makes it easy to write and debug codes.

3) REINFORCEMENT LEARNING

Many tools for RL are built on top of classical ML and deep learning tools, to support various algorithms.

[25] https://github.com/JayWalker512/PacketGAN
[26] https://scikit-learn.org/stable/
[27] https://numpy.org
[28] https://scipy.org
[29] https://keras.io/
[30] https://www.tensorflow.org/
[31] https://pytorch.org/

OpenAI Gym / Gymnasium: OpenAI Gym[32] (lately continued as Gymnasium[33] by the Farama Foundation) is an open-source Python library that provides a standardized API for the interaction between RL algorithms and environments. Additionally, it includes a wide range of environments of different complexities, including classic control tasks, Atari games, robotic simulations, as well as physical simulations. This allows researchers to reproducibly benchmark RL algorithms on a standardized set of environments. Furthermore, Gym can be extended by custom environments, allowing users to easily compare the performance of different RL algorithms for customized problems.

One challenge of RL research is that different implementations of the same RL algorithm can have significantly different performances in the same environment, making RL algorithms highly sensitive not only to hyperparameters but also to small implementation details [186].

Stable Baselines3: Stable Baselines3 (SB3) [187] is an open-source Python library that contains reference implementations of seven widely used DRL algorithms. Tab. 10 lists all supported algorithms. The performance of those algorithms has been thoroughly tested. The library is compatible with the OpenAI Gym/Gymnasium API, enabling users to train RL agents in just a few lines of code. Moreover, the library supports custom Gym environments, custom policies for the algorithms, TensorBoard, as well as data logging customization through custom callbacks.

Additional RL algorithms are implemented in the Stable Baselines3 Contrib (SB3-Contrib)[34] package. These are implementations of newly published algorithms. They are less tested and therefore considered experimental.

RL Baselines3 Zoo: RL Baselines3 Zoo[35] is a Python library that provides pre-trained agents and a set of optimized hyperparameters for the algorithms from SB3 and the Gym environments. Moreover, it provides useful helper scripts for training and evaluating agents, for tuning hyperparameters, and for plotting results.

CleanRL: CleanRL [188] is a DRL framework that provides thoroughly benchmarked single-file Python implementations of eight DRL algorithms (c.f. Tab. 10). Its goal is to provide researchers full control over an algorithm in a single file, making it easier to 1) fully understand all implementation details, and 2) quickly prototype novel DRL features. In addition, it provides support for TensorBoard. In comparison to SB3, CleanRL does not provide a high-level user-friendly API for model training. It is instead tailored to provide a development environment for DRL researchers with implementations that are easy to read, debug, modify, and study. The desired workflow is to first prototype new RL ideas in CleanRL and afterwards port it to a library offering a higher-level API like SB3.

OpenAI SpinningUp: OpenAI SpinningUp[36] is a great resource for aspiring researchers and practitioners that are excited to apply DRL to their problems but are overwhelmed by the implementation complexity of algorithms in frameworks like Stable Baselines3. It provides detailed explanations of the most important concepts of DRL, as well as explanations and implementations of key DRL algorithms. The algorithm implementations specifically focus on simplicity with the aim of being easy to follow for people new to the field. This simplicity is achieved by narrowing down the implementations to the core concepts of the algorithms, and by omitting more complex features that can significantly improve the algorithm's performance. As a result, OpenAI SpinningUP should be primarily seen as a resource for education that should not be used in production systems.

PettingZoo: PettingZoo[37] is an open-source Python library that contains a set of environments for multi-agent reinforcement learning. While it is similar to OpenAI Gym/Gymnasium in its functionality and API, the application scenario of MARL is different from the one of single-agent RL. Among others, it contains multi-agent environments of Atari games and classic games like chess and Go. Furthermore, it can be extended by custom environments.

Ray RLlib: Ray RLlib [189] is an open-source Python library for RL. Out of the RL libraries presented in this section, it is the most comprehensive one. It supports a wide range of performance-tested RL algorithms, offers a high-level user-friendly API to train agents, supports single-agent, multi-agent, and custom environments, offers high scalability by supporting both single-machine and distributed training, and offers tools for managing, tracking, and visualizing the results of experiments. Because it is built on the Ray platform, it is also seamlessly compatible with other Ray libraries and tools for distributed computing and parameter tuning.

When to use which RL library? An important question to answer in this primer is *when to use which of the presented RL libraries? CleanRL* is recommended to be used either to fully understand how an algorithm is implemented or by RL researchers to quickly prototype new ideas, since its design decision to separate each algorithm into its own file lets the researcher focus on the algorithm instead of the complex software architecture of other RL algorithm libraries with intertwined modular implementations. *SB3* is primarily intended to offer well-tested baseline implementations of important DRL algorithms as a benchmark baseline for new RL developments. However, along with its extensions *SB3-contrib* and *Zoo*, it is recommended to be used if a high-level interface for fast training of well-established and well-tested RL algorithms on single-agent environments are desired and no scalability via distributed learning is required. *RLlib* offers a production-ready framework for large scale projects. It is recommended to be used for multi-agent environments, as well as when high scalability via distributed learning,

[32] https://www.gymlibrary.dev
[33] https://gymnasium.farama.org
[34] https://sb3-contrib.readthedocs.io/en/master/
[35] https://github.com/DLR-RM/rl-baselines3-zoo
[36] https://spinningup.openai.com
[37] https://pettingzoo.farama.org

TABLE 10. Overview of the implemented single-agent algorithms of different RL frameworks.

	SB3	SB3 Con.	CleanRL	Spin.UP	RLlib
A2C [52]	+	-	-	-	+
DDPG [110]	+	-	+	+	+
DQN [109]	+	-	+	-	+
HER [188]	+	-	-	-	-
PPO [189]	+	-	+	+	+
SAC [190]	+	-	+	+	+
TD3 [191]	+	-	+	+	+
ARS [192]	-	+	-	-	+
QR-DQN [193]	-	+	-	-	-
RecurrentPPO [194]	-	+	-	-	-
TQC [195]	-	+	-	-	-
TRPO [196]	-	+	-	+	-
Maskable PPO [197]	-	+	-	-	-
Categorical DQN [198]	-	-	+	-	-
PPG [199]	-	-	+	-	-
RND [58]	-	-	+	-	-
VPG [49]	-	-	-	+	+
A3C [52]	-	-	-	-	+
AlphaZero [200]	-	-	-	-	+
Behavior Cloning [201]	-	-	-	-	+
CQL [202]	-	-	-	-	+
CRR [203]	-	-	-	-	+
Dreamer [204]	-	-	-	-	+
IMPALA [205]	-	-	-	-	+
R2D2 [206]	-	-	-	-	+
Rainbow [207]	-	-	-	-	+
SlateQ [208]	-	-	-	-	+
DD-PPO [209]	-	-	-	-	+

e.g., on clusters, is required. Furthermore, RLlib includes tested implementations of cutting-edge RL algorithms. Concerning environments and the interaction of agents with them, *Gymnasium* and *PettingZoo* are arguably the most recognized and thus important standard APIs for single-agent and multi-agent RL, respectively. Creating environment interfaces that adhere to their API definitions makes it much easier to experiment with different RL algorithms and variations, since many implementations expect Gymnasium or PettingZoo environment instances.

Table 10 provides an overview of the algorithms implemented in the different RL frameworks. Besides the listed algorithms, RLlib supports further algorithms specifically for MARL.

D. DATA LOGGING AND PARAMETER TUNING

Prototyping is an essential aspect of ML development, and the ability to log and monitor experiments is crucial for efficient iteration. Creating individual names for logs and artifacts might work after ten runs but becomes overwhelming after hundreds of runs. When trying to compare different runs or when looking for the respective parameters of a run, having to look into log files is cumbersome. Fortunately, a range of tools has been developed to do this task, organizing every run with its parameters, visualizing runs with plots and the option to filter and search for specific runs, which will be introduced in this section. These tools provide a suite of features beyond just logging and monitoring, including the ability to perform hyperparameter tuning. Additionally, most of them easily integrate with popular ML frameworks such as TensorFlow, PyTorch, Keras, and Scikit-learn.

1) DATA LOGGING

TensorBoard is by far the most popular data logging and visualization tool in ML. It is widely used in conjunction with deep learning frameworks like TensorFlow and PyTorch. However, there are other popular data logging and visualization tools as well, such as Weights & Biases (WandB) and Comet.ml. The popularity of these tools may vary depending on the specific use case and developer preferences. We present in Table 11 a list of the most popular tools. These tools all support different ML frameworks and offer real-time monitoring and visualization of ML models. They also handle various data types and provide customizable views. Weights & Biases and Comet.ml provide experiment tracking and collaboration features that allow for easy collaboration and sharing of results, while Visdom and TensorWatch are great options for debugging and developing since they offer configurable logging options and real-time tensor viewing.

It's important to keep in mind that the developer's individual demands and preferences will determine the visualization tool that fits best. Some developers might favor a tool that integrates well with their preferred ML framework, whereas others would value flexibility and customization possibilities. Also, some developers could need more sophisticated features like collaboration tools or hyperparameter optimization, while others might only need the most fundamental logging and visualization options.

2) TUNING TOOLS

In ML, hyperparameters are often used to control the behavior of the ML model. These hyperparameters include the usual parameters for training such as batch size, which optimizer to use, learning rate and other optimizer specific parameters, but can also include the structure of the model like number, type and topology of layers, or other custom parameters specific to the application. Selecting optimal hyperparameters is critical to achieving the best possible performance of a ML model. However, searching for the optimal hyperparameters can be a complex and time-consuming task, especially for large datasets and complex models. Tuning tools help to automate this process by systematically searching for optimal hyperparameters based on a user-defined search space and optimization criteria. This can save significant time and resources in the ML development process and help to achieve better model performance. Below is a list of the most popular tools:

- *NNI* (Neural Network Intelligence) [218]: an open-source toolkit developed by Microsoft for automating and optimizing the hyperparameter tuning process of deep learning models. It provides a framework for designing and conducting experiments using various search algorithms and techniques, such as grid search, random search, Bayesian optimization, evolutionary optimization, and tree-structured Parzen estimator (TPE). It also supports distributed training and can scale up to thousands of nodes for high-performance computing.

TABLE 11. List of Visualization and Logging Tools.

Tool	Features and Capabilities	Integration with ML Frameworks	Additional Features
TensorBoard [210]	Web-based visualization tool for monitoring and analyzing ML models. Features include histograms of weights and biases, visualization of the computational graph, and tools for comparing models.	TensorFlow and Pytorch	-
Weights & Biases [211]	Platform for ML experimentation, including experiment tracking, visualization, and collaboration. Provides monitoring of metrics, model performance, and hyperparameter optimization via sweeps.	TensorFlow, PyTorch, Keras, XGBoost, and more	Experiment tracking and collaboration tools
Comet.ml [212]	Platform for ML experimentation, including experiment tracking, visualization, and collaboration. Features include experiment management, collaborative workspace, and model versioning.	TensorFlow, PyTorch, Keras, Scikit-Learn, and more	Experiment tracking and collaboration tools
MLflow [213]	Open-source platform for ML experimentation, including experiment tracking, visualization, and model deployment. Features include experiment management, model registry, and reproducible code management.	TensorFlow, PyTorch, Keras, Scikit-Learn, and more	Experiment tracking and model management
Visdom [214]	Web-based visualization tool for deep learning. Provides real-time plotting of various types of data, including images, video, and text, and supports PyTorch. Features include custom visualizations, support for multiple windows, and flexible logging.	PyTorch	Flexible logging and custom visualizations
TensorWatch [215]	Debugging and visualization tool for deep learning. Allows visualization of data flow graphs, histograms, and confusion matrices, and supports TensorFlow, PyTorch, and MXNet. Features include real-time inspection of tensors, automatic model visualization, and integration with Jupyter Notebook.	TensorFlow, PyTorch, MXNet	Real-time tensor inspection and automatic model visualization

- *Optuna* [219]: an open-source hyperparameter optimization framework for ML. It provides a flexible and modular platform for automating the process of selecting optimal hyperparameters for a given model architecture. Optuna uses various algorithms to search the hyperparameter space, including TPE, Covariance Matrix Adaptation Evolution Strategy (CMA-ES), Non-Dominated Sorting Genetic Algorithm II (NSGA-II), and adaptive sampling. It also supports distributed optimization across multiple nodes for faster and more efficient tuning.
- *Ray Tune* [220]: the hyperparameter tuning component of the Ray framework. It handles the execution of experiments including parameter studies with possibly multiple repetitions as well as scheduling the runs for parallel execution. For hyperparameter tuning, it supports a wide variety of approaches. These include basic strategies such as grid or random search, but also more advanced approaches such as Bayesian optimization or Population Based Training [221]. While some algorithms are implemented internally, it relies heavily on third-party optimization libraries such as Hyperopt [222] and Optuna [219], and provides a unified interface to them.
- Keras Tuner [223]: a library customized for Keras that provides an easy-to-use API for defining a hyperparameter search space, choosing search algorithms such as random search and Bayesian optimization, and running hyperparameter search processes. Furthermore, Keras Tuner is easy to integrate with other Keras workflows and can optimize both single-node and distributed hyperparameters.
- Hyperopt [224]: a Python library for hyperparameter optimization that uses a combination of random search and Bayesian optimization to efficiently explore and exploit the hyperparameter search space. It provides an easy-to-use API for defining the hyperparameter search space, selecting optimization algorithms, and executing the hyperparameter search process. Hyperopt uses a Tree-structured Parzen Estimator (TPE) algorithm to model the relationship between hyperparameters and model performance and to guide the search for better hyperparameters. Hyperopt also allows for the parallelization of the search process, making it scalable to large hyperparameter search spaces and parallel computing environments. It can be used with a variety of machine-learning frameworks, including Scikit-learn, Keras, and PyTorch.

- Scikit-Optimize [225]: a Python library for sequential model-based optimization that aims to efficiently explore and exploit the hyperparameter search space while minimizing the number of model evaluations. It provides a simple and flexible API for defining the hyperparameter search space and selecting optimization algorithms, including Bayesian optimization and gradient-based optimization. Scikit-Optimize also supports parallel evaluation of the search process, making it scalable to large hyperparameter search spaces and parallel computing environments. In addition to hyperparameter optimization, Additionally, it can be used for function optimization and global optimization tasks. Furthermore, it integrates easily with popular ML frameworks such as Scikit-learn and Keras, while including features such as early stopping and warm-starting to further improve the efficiency of the hyperparameter search process.

Note that Table 12 presents only the most commonly used algorithms for each tool. While other algorithms may be added, the mileage may vary depending on the specific use case and requirements. Overall, the choice of which tool to use depends on the specific requirements and use case. For example, if there is a need for scalability and distributed training, Ray Tune is a good choice. If there is a need for a general-purpose optimization library, then Scikit-Optimize might be a good choice.

E. TESTBEDS

As previously outlined, it is hard to replicate realistic network conditions, and using existing datasets might not always fit the problem. While simulation tools can help with that, there is also the possibility of using existing testbeds or building your own. Access is usually open or free to researchers for the existing real-world testbeds, but you might have to schedule your experiments and wait depending on utilization. In the following, some popular real-world testbeds and some devices one could use to build a testbed will be introduced. There are two types of testbeds, wired ones, and wireless ones. The wireless ones are wireless sensor networks without any routers or switches, and communication is broadcasted. Testbeds are relatively versatile and can be used to either test ML applications that rely on networks like Distributed ML or to test ML algorithms that do traffic routing, for example.

1) REAL-WORLD TESTBEDS

For a more extensive overview, [226], [227], [228], and [229] provide surveys that either include a section about testbeds or are entirely about testbeds. We present a selection of popular testbeds, starting with wireless testbeds.

FlockLab [230] is an experimental platform that enables researchers to test and evaluate the performance of wireless sensor networks (WSN) and IoT systems. It is a flexible, open-source testbed that provides a controlled and repeatable environment for the evaluation of various applications. An advantage of FlockLab is its flexibility, as it can be used to test and evaluate a wide range of wireless sensor networks and IoT systems [230]. It supports various wireless technologies, such as Zigbee, Z-Wave, and LoRaWAN, and it can be easily extended to support new technologies. FlockLab is widely used in the field of WSNs and IoT systems [231], and it has been developed and maintained by the Communication Systems Group at ETH Zurich.

FIT IoT Lab [232] is an open-access testbed for IoT experiments provided by the French Institute of Technology. It contains over 1500 nodes offering a wide range of low-power wireless devices that can be used to test and evaluate various IoT applications, protocols, and algorithms. In addition, its large-scale infrastructure and easy-to-use web interface provide a flexible and convenient platform for IoT experimentation.

D-Cube [233] is a testbed by Graz University of Technology. It contains about 50 nodes with two platforms, nRF52840 and TelosB, and provides a set of predefined scenarios. These scenarios allow researchers to evaluate protocol performance and compare it against each other easily.

CLOVES [234] is a part of the IoT Testbed at the University of Trento. It contains 275 indoor devices spread over 8000 square meters. Communication is possible using ultra-wideband or narrowband, and all nodes are remotely accessible.

Next, we are going to introduce some wired testbeds. Note that some of them also provide wireless capabilities.

PlanetLab [235] was founded in 2002 by researchers from several universities, including Princeton University, the University of California at Berkeley, and Stanford University. While it was shut down in March 2020, *PlanetLab Europe*[38] continues to operate. It is a collection of interconnected computers located at over 250 sites in more than 40 countries across Europe and beyond, available for researchers to use in their experiments. *PlanetLab Europe* provides researchers with virtual machines, storage, and network connectivity. In addition, researchers can deploy their software on the nodes and create custom network topologies to simulate various network scenarios.

EmuLab[39] [236] is a network testbed developed by the University of Utah that provides users with a virtual network environment to test and evaluate various networking systems and applications. Emulab allows researchers to create and configure network topologies, deploy software and network services, and generate different types of network traffic to test and evaluate various networking scenarios.

GENI (Global Environment for Network Innovations) [237] is a US national-scale network testbed that provides researchers with a virtual laboratory for developing and testing new networking technologies and applications. It comprises a large-scale network of interconnected computing

[38] https://www.planet-lab.eu/
[39] https://www.emulab.net/

TABLE 12. Comparison of Hyperparameter Tuning Tools.

Tool	Common Algorithms	ML Libraries	Scalability	Ease of Use	Community
Ray Tune	Hyperband, PBT, Bayesian	TensorFlow, PyTorch, others	High	Simple API	Large
Hyperopt	Bayesian	TensorFlow, PyTorch, others	Medium	User-friendly	Large
Optuna	TPE, CMA-ES, NSGA-II	TensorFlow, PyTorch, others	Medium	User-friendly	Large
Keras Tuner	Random search, Bayesian, Hyperband	Keras	Low	Very user-friendly	Small
Scikit-Optimize	Bayesian, gradient-based	TensorFlow, PyTorch, others	Low	Requires programming experience	Small
NNI	Grid search, random search, evolutionary optimization, TPE	TensorFlow, PyTorch, MXNet	High	Simple API	Large

resources, including servers, routers, switches, and other network devices.

2) BUILDING YOUR OWN TESTBED

When seeking greater control over a testbed, building a customized one emerges as a viable option. Fortunately, there are several cost-effective devices available for this purpose, with some even incorporating machine learning accelerators [238]. These accelerators enable the deployment of machine learning models for training and inference within the testbed and offer a variety of communication approaches. In this section, we will provide a list of the most common and popular devices used for this purpose, along with detailed explanations of their respective advantages.

NVIDIA Jetson[40] is a series of embedded computing boards designed for IoT and ML applications. They include NVIDIA GPUs and CPUs, as well as a variety of interfaces and sensors for connecting to other devices. Jetson boards are designed to be low-power and compact, making them suitable for portable and battery-powered applications. They can be used for various tasks, including image and video processing, deep learning, and robotic control.

Google Coral[41] includes a range of hardware and software products, such as the Coral Dev Board, the Coral USB Accelerator, and the Edge TPU software. The Coral Dev Board is a single-board computer that is designed to be small and low-power, making it suitable for use in portable and battery-powered devices. It has a system-on-a-chip (SoC) that includes a Google Edge TPU, which is a custom-built Tensor Processing Unit (TPU) for running ML/DL models. The Coral USB Accelerator is a small USB device that can add Edge TPU capabilities to existing devices. The Edge TPU software provides a set of libraries and tools for developing and deploying ML models.

The *Raspberry Pi* boards are equipped with a variety of interfaces and peripherals, such as USB ports, Ethernet, HDMI, and a 40-pin expansion header. They also have high CPU and memory capacity, which makes them powerful enough to run various applications. The Raspberry Pi can run TensorFlow Lite and other ML frameworks, enabling researchers to run pre-trained models and perform basic ML tasks. It can also be used as an edge device for collecting and preprocessing data before sending it to the cloud for further analysis.

The *Intel Movidius Neural Compute Stick*[42] is a USB device that provides on-device AI inference for various applications in networked systems. It features a Myriad 2 VPU, which can run deep neural networks with low power consumption. The Neural Compute Stick can accelerate computer vision, speech recognition, and natural language processing tasks in networked devices.

V. EXPLAINABLE ARTIFICIAL INTELLIGENCE

While ML and especially DL models are powerful tools for network service providers, they come with the major drawback that their reasoning is difficult to understand for humans due to their black-box characteristics [239]. This lack of understanding may result in stakeholders, e.g., network service providers, not deploying ML models in production environments as they do not trust their reasoning and, thus, fear outages or revenue losses. To alleviate these concerns, Explainable AI (XAI) is well-suited as it helps to understand the underlying reasoning of ML models. This reasoning is achieved by intelligently relating inputs and outputs. The thereby learned transformation function or only some parts of it become interpretable. Usually, this interpretability comes in the form of mathematical functions or as heatmaps describing the influence of the inputs on the model's decision. In addition, a quantification of a model's uncertainty is fundamental for risk assessment during deployment, thereby paving the way for Responsible AI.

There are plenty of use cases to apply XAI in communication networks [240]. These use cases include network planning and engineering [241], resource allocation [242], [243], performance management [128], [244], and security

[40] https://www.nvidia.com/de-de/autonomous-machines/embedded-systems/

[41] https://coral.ai/

[42] https://www.intel.com/content/www/us/en/developer/articles/tool/neural-compute-stick

management [245], [246]. Most of these works use the methods presented in this chapter to make their models explainable.

A. TAXONOMY OF XAI METHODS

A general overview of XAI techniques is provided in [247] and an extensive survey on XAI methods as well as a taxonomy for XAI methods in general can be found in [248]. XAI methods can be classified into techniques which explain a model locally or globally. A local explanation technique provides model explanations for a single input, e.g., why is a specific packet routed that way, while a global explanation technique provides general explanation strategies of a model, e.g., how does the model route packets in general.

Further, XAI methods can be classified into post-hoc explainers and interpretable models. Post-hoc explainers are utilized to explain various already trained black-box models, e.g., neural networks or ensemble models. Ensemble models like Random Forest are composed of multiple smaller models jointly determining the output. This makes interpretation difficult. Interpretable, transparent, or glass-box models provide an explanation for how the model obtains the output by design. Prevalent models are, for example, the well-known linear models and decision trees, as well as the less-known generalized additive models.

Finally, model-agnostic methods and model-specific methods are distinguished. Model-agnostic methods can be used on top of every kind of model, while model-specific methods can only be used by specific model families. A prominent example of model-specific methods are saliency maps [249], which are computed from the feature maps learned by a model and can be used in computer vision to highlight the regions on which the model focuses when processing input. They are generally applicable when using CNNs. This also implies that the nature of the data directly influences the applicable XAI techniques for the different use cases, e.g., time series XAI techniques are not usable for graph data.

B. SPECIFIC XAI METHODS

Since there are many different categories of XAI techniques, there is a wide spectrum of specific XAI methods. Thus, the following explained methods are only a small selection. Due to the fact that in many XAI scenarios a black-box ML model should be intelligible, the methods introduced first focus on post-hoc explainer. While it is common to perform post-hoc explanations, the authors of [250] argue that we should stop using post-hoc explainers and instead directly use interpretable models. Interpretable models often perform weaker than black-box models, but are interpretable by design.

1) POST-HOC EXPLAINERS

As a majority of advances in ML happen in computer vision, there exists a huge variety of post-hoc explainers explaining the learnt filters of a CNN, e.g., saliency maps. As a consequence, these techniques are model-specific and usually not applicable for network data. Nevertheless, there exist approaches where network data is transformed to images beforehand, e.g., for encrypted network traffic classification in [251], and processed with a CNN, so saliency maps could be applied here.

Layer-wise Relevance Propagation (LRP) is a post-hoc method that uses the neural network's forward pass and propagates its output backwards through the layers until the input layer to derive the relevance of an input on the model's prediction.

A prevalent local model-agnostic post-hoc explainer is called SHapley Additive exPlanations (SHAP), which uses methods from game theory to judge the importance of different feature inputs. Although this method can explain the black box of a ML model very well, it comes with the drawback that it needs high computational power. Thus, it is only feasible for models with fewer input parameters [252].

A well-working method for getting an explanation of classification models in a model-agnostic fashion is a method named Local Interpretable Model-agnostic Explanations (LIME) [253]. LIME belongs to the class of surrogate models, where a model is used to approximate the predictions of a target black-box model to infer the reasoning of the black-box model. LIME trains a local surrogate model to explain the predictions for a specific sample by first aggregating permutations of the original feature inputs of the sample into a new dataset, weighting the samples of the dataset according to their proximity to the original sample, and then training an interpretable model on this dataset to approximate the predictions of the black-box model. After training, the local model can be interpreted to understand the black-box model's reasoning.

Another type of local model-agnostic post-hoc explainers are counterfactual explanations [247]. Counterfactual explanations are used for causal reasoning and may serve to answer what-if questions, i.e., "would Y have occurred if X had not occurred before". These techniques may be helpful for network operators when they try to analyze and manage their network with respect to critical situations, e.g., how to avoid congestion in a network. In a nutshell, they work by deriving causal relationships from the input features and then manipulating input features to perform specific reasoning.

2) INTERPRETABLE MODELS

The easiest to interpret and most known interpretable models include decision trees, which are interpretable in an if-else fashion, and linear models like linear regression or logistic regression, where slope and intercept directly characterize the input mapping. Generalized linear models (GLMs) and generalized additive models (GAMs) extend linear models to better reflect non-linear functions and different target distributions other than Gaussian distributions as with linear regression [247]. Especially for GAMs, many different models exist by now that are directly interpretable.

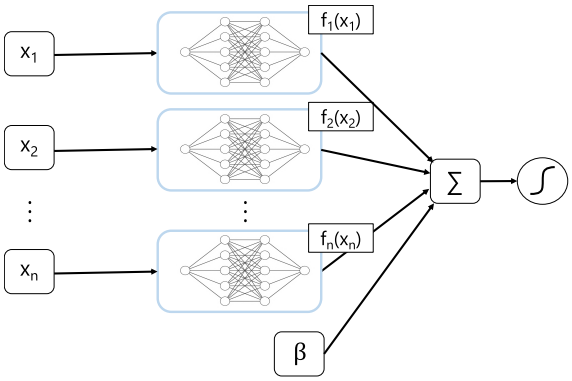


FIGURE 7. Architecture for Neural Additive Model (NAM) [255].

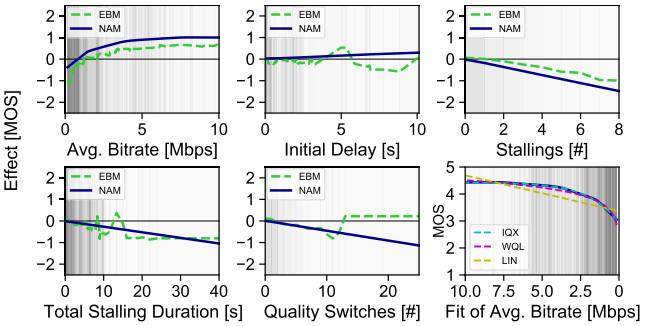


FIGURE 8. Interpretability of EBM and NAM for an exemplary video streaming QoE modelling scenario [256].

The main idea behind GAMs is that a separate function f_i is learnt for each feature x_i and that the outputs of these functions are then summed up with a bias β to perform the prediction. The bias can hereby be learnt or fixed beforehand. Variants of GAM include the Explainable Boosting Machine (EBM) [254], which is a tree-based GAM and uses gradient boosting to improve performance, and the Neural Additive Model (NAM) [255], which consists of n neural subnetworks (f_i) transforming each of the n input features to a new representation. The architecture of NAM with its n subnetworks and the bias β is depicted in Fig. 7.

In Fig. 8, we show the interpretability of EBM and NAM with the exemplary use case of Quality of Experience (QoE) modelling for video streaming [256]. QoE describes the delight or annoyance of a user with a networked service as influenced by different factors, e.g., context factors or system factors [257]. In order to avoid customer churn, network service providers and ISPs are thus interested in estimating QoE or, as a proxy, QoE influence factors from encrypted network traffic. For this purpose, most approaches rely on ML as can be seen for video streaming and web browsing in [157] and [258]. For data-driven QoE modelling, the goal is to predict the Mean Opinion Score (MOS) of a service on the range from 1 (bad) to 5 (excellent) in a regression task given some QoE influence factors. With video streaming QoE, some of these influence factors include video quality, initial delay, and stalling events (interruption of playback due to buffer underrun), and also quality switches. Given a sample X with the individual feature values x_1 to x_5 for avg. bitrate, initial delay, number of stallings, total stalling duration, and number of quality switches, the model simply transforms each feature input x_i with the learnt function f_i to an effect and sums up these effects along with the bias β to obtain the prediction. The figure for example shows that an increase in avg. bitrate leads to a higher MOS (positive effect) and that stalling events severely degrade the MOS (negative effect). A service provider should therefore try to increase the avg. bitrate and try to avoid stalling events to improve QoE. The learnt functions of the model can be easily interpreted and the model's decision can therefore always be reconstructed from the inputs. Given some experts' domain knowledge, the learnt functions can also be verified with the expected experts' functions. Both these aspects strongly increase the trust towards the model and, thus, allow actual deployment.

C. UNCERTAINTY

One major issue with every form of ML is the fact that training usually consists of fitting models such that the mean response of the model approximates the groundtruth in a best possible way. This results in the model learning point estimates, which may be close to the groundtruth or very far away from it for single data points, but across all data points the model may be performing well on average. While this may be acceptable for some use cases, for other use cases this becomes problematically as soon as wrong decisions become costly for stakeholders, e.g., a ML model organising the routing tables in a network may cause downtimes in a network due to wrongly deleting or adding routing rules. Thus, stakeholders have to assume that the model does not always perform as expected. This, however, impedes the deployment of ML models in general, and also in particular in communication networks, and a way to quantify the certainty or, more important, the uncertainty is required.

Uncertainty thus describes a model's insecurity about its prediction. Uncertainty can be divided into aleatoric uncertainty and epistemic uncertainty [259]. Aleatoric uncertainty resembles the uncertainty in the data used for training and testing. This kind of uncertainty can never be reduced as the underlying process of the training and test data generation already includes noise. Epistemic uncertainty, on the other hand, can be reduced as this uncertainty is caused by the ML model due to a lack of training data at specific points or an insufficient capacity, i.e., the model is not complex enough to capture the actual relationships between the features. Hence, epistemic uncertainty can also be considered as model uncertainty. This uncertainty can be reduced by collecting for example more training data for training points, where data is scarce, or by increasing model complexity.

In most ML and XAI models, a way to quantify uncertainty is usually not included. Instead, different means to quantify aleatoric and epistemic uncertainty have to be used. Note also

TABLE 13. Overview on XAI techniques and libraries.

	Model	Local	Global	Classif.	Regr.	Libraries
Post-Hoc	SHAP	✓	✗	✓	✓	SHAP, InterpretML, OmniXAI, Alibi
	LIME	✓	✗	✓	✗	LIME, InterpretML, OmniXAI, ELI5
	Anchors	✓	✗	✓	✗	Anchors, Alibi
	Gradient-based Methods	✓	✗	✓	✓	OmniXAI, Alibi, Captum, TorchRay
	Permutation Importance	✓	✗	✓	✓	OmniXAI, ELI5, Alibi
	Partial Dependence Plot	✓	✗	✓	✓	Scikit-learn, OmniXAI, Alibi, explainX
Interpretable	Linear Regression	✓	✓	✗	✓	Scikit-learn, InterpretML, OmniXAI
	Logistic Regression	✓	✓	✓	✗	Scikit-learn, InterpretML, OmniXAI
	Decision Tree	✓	✓	✓	✓	Scikit-learn, InterpretML, OmniXAI
	Decision Rules	✓	✓	✓	✓	InterpretML, OmniXAI
	Explainable Boosting Machine (EBM)	✓	✓	✓	✓	InterpretML
	Neural Additive Models (NAM)	✓	✓	✗	✓	Github
	TabNet	✓	✗	✓	✓	TabNet

that many approaches quantify only epistemic or aleatoric uncertainty, but not both simultaneously. A survey over existing approaches is provided in [260]. In the following, some selected ways to quantify uncertainty are shortly introduced. A way to estimate epistemic uncertainty is for example the use of ensembles, i.e., training the same model with different seeds and considering the predictions of each model and, in particular, the differences in these predictions. The stronger the differences between the models' predictions, the higher the uncertainty. This approach can be used for any kind of model. Another simple approach for estimating epistemic uncertainty in neural networks is the use of Monte Carlo Dropout. With Monte Carlo Dropout, the dropout layers, which are usually used for improved model generalizability during training, are also kept active during inference. Generating multiple model predictions with active dropout can also be considered as approximate Bayesian inference. Again, the variation in the returned predictions quantifies the degree of uncertainty. To learn aleatoric uncertainty, it is usually required to learn in the model not only mean responses, but instead the variance must be learnt, too [261]. With neural networks and a regression task, this is for example easily possible by simply adding another head, i.e., output neuron, to the neural network, which learns the variance and by accordingly adjusting the loss function. Using the negative log-likelihood of a Normal distribution (or any other distribution) as loss function, it is thus for example possible to learn a Normal distribution for an input, thereby allowing to quantify the uncertainty in form of the variance for an input. Finally, Bayesian Neural Networks as proposed by Kendall and Gal [261] can model both aleatoric and epistemic uncertainty. With Bayesian Neural Networks, model weights are assigned a probability distribution instead of a single value. Using these probability distributions, it is then possible to quantify epistemic uncertainty. For aleatoric uncertainty, they simply use two heads, where they learn both mean and variance for a data point.

D. RESPONSIBLE AI

Strongly related to uncertainty is the concept of Responsible AI. According to Arrieta et al. [248], XAI alone is not sufficient for an ethical and responsible usage of ML models. Responsible AI is in general a much broader topic than XAI [262]. With responsible AI, there are additional principles, which must be kept in mind, when developing and deploying ML models. These principles include the prevention of discrimination against persons, groups, or races, i.e., the model must be fair. In the context of communication networks, this could for example mean that a model discriminates specific users by assigning them lower bandwidth shares and a higher latency. Additionally, responsible AI ensures that users or stakeholders are always aware of the usage of ML models. Specifically, it must be transparent to everybody that ML has been used and how it has been used. An example for communication networks is the adaptive change of a routing table by an ML model. The model must be able to outline why a change was required and why it has changed specific routes. Next, the use of ML models should always end up in a beneficial way for humanity in all aspects of life. They should not be used in disruptive ways, e.g., generating downtimes in a network for specific users on purpose. Finally, privacy and security is also a very important topic. ML models require data. Here, the privacy and security of sensitive data must be kept throughout the whole lifecycle of preparing and deploying the model. Responsible AI is still a young field of research. Nevertheless, all the mentioned principles must be kept in mind when preparing and deploying ML models in practice. It is one of those topics already diligently discussed in the conceptualization of future networks, e.g., 6G [263]. Meanwhile [264] is a more generic survey of best practices to ensure that the AI environments are responsible.

E. LIBRARIES

Several XAI libraries are available for all kinds of frameworks, e.g., Scikit-learn, PyTorch, and TensorFlow (cf. Section IV). Microsoft created a Python library named InterpretML [254], which unifies black-box explainers, e.g., SHAP values, LIME, or Partial Dependence Plots, and transparent models, e.g., linear models, decision trees, decision rules, and also EBM, a tree-based generalized additive model. OmniXAI [265], AIX360 [266], and Alibi [267]

also provide a collection of various post-hoc explainers and models for all kinds of data types and backends. In contrast to the libraries containing several different tools, individual explainers like SHAP or Anchors, but also interpretable models like the attention-based model TabNet[43] are available as separate Python packages.

All gradient-based methods, e.g., Integrated Gradients [268], can be directly implemented within PyTorch and TensorFlow, or additional libraries like Captum [269], TorchRay [270] and TF-Explain [271] can be used. Captum also comprises a huge number of techniques for explaining image-based data.

VI. NETWORKS FOR MACHINE LEARNING

In the previous sections, we primarily explained ML methods, architectures, and principles to develop ML models. Hence, we focused on applying ML to design and optimize networks, detect patterns and anomalies, and predict network behavior autonomously. We refer to this application as ''ML for Networks'' [272], [273], where ML models are developed from network data to, e.g., design the communication topology of a network or to balance the traffic load.

However, networks and ML form a mutual relationship in which networks support ML, e.g., by using a network as an infrastructure for ML algorithms, both for training and inference. As we will see throughout this section, networks are thus a key success factor for ML by connecting and providing computational power and data storage [274], [275]. We refer to this support and infrastructure functionality of networks as ''Networks for ML''. Important to note is that it is detached from the ML model application. Instead, any ML model can be trained or deployed in a networked system. As ML is a relatively new network task, challenges for networks arising from ML traffic and possible effects on ML from networks are still the subject of research. ''Networks for ML'' generally comprises these open research questions.

ML algorithms primarily use a network to access data from memory or to exchange model parameters/updates. The traffic load generated, the traffic shape, and network requirements, e.g., regarding latency and robustness, are unknown for many ML methods and are likely to be application-specific and method-specific. All this can pose new challenges for networks and make a better understanding of the mutual relationship between ML and networks necessary. Thus, it is no longer sufficient to evaluate ML model performance alone but also the network performance. Hence, one might ask the question: Which metrics to use to evaluate model and network performance when applying ML in networks?

From Section II, we know, that several metrics can be used to evaluate the performance of ML models. These metrics could depend on the specific task or application of the model [276]. Although these metrics were introduced for ML models with network application (''ML for Networks''), it is worth noting that some metrics can also help answer the questions arising in ''Networks for ML''. The choice of metrics will depend on the specific problem and the desired outcome. Hence, ''ML for Networks'' and ''Networks for ML'' are not mutually exclusive [277].

For instance, Data Quality is a metric that can be used for evaluating both. As ML is generally data-driven, data quality is very important for model development. Thus, when ML is applied for network tasks, data quality is often determined primarily measured by the correctness and representativeness of events/classes. This can also be utilized in ''Networks for ML''. However, as it focuses on decentralized data sources, data distribution can additionally be considered. Other metrics typically considered by the ML community are: Privacy, Robustness, Energy Efficiency, and Fairness. However, as ''Networks for ML'' also focuses on network behavior, typical network metrics are often applied, such as Throughput, Latency, Packet loss rate, and Spectral efficiency. Table 14 further explains the metrics and their impact.

So why do these network metrics influence the ML models? High latency and low throughput (as well as low spectral efficiency) can cause delays in the training process, leading to slower training times and increased iteration cycles. Packet loss can impact the accuracy and also the consistency of ML models, because it can lead to incorrect or incomplete data inputs, and can cause inconsistent data transfer in case of retransmissions. This, in turn, can affect the model's ability to generalize, converge and make accurate predictions.

Different network topologies could affect the ''Networks for ML'' performance, scalability, and security. Furthermore, when considering ML for Networks, the choice of network topology can also affect the accuracy and efficiency of the models.

For example, in a star topology, all nodes are directly connected to a central hub, which can make the network easier to manage and administer. From a ML perspective, this topology would lend itself to centralized learning, where data from all nodes is collected and processed in a central location. This approach could *simplify the deployment and maintenance* of the ML model, but it could also lead to a single point of failure and potential privacy concerns.

On the other hand, a mesh topology, in which nodes are connected in a decentralized fashion, can be more *resilient to failures* and provide more privacy, but it can also be more difficult to manage. In terms of ML, this topology can be suitable for distributed learning, where each node trains a local model and shares its knowledge with the other nodes. This approach could *improve the scalability* and *privacy* of the model, but it could also increase the synchronization overhead.

There are also other network topologies, such as bus, ring, tree, and hybrid, which can have different tradeoffs in terms of network metrics. Choosing the right topology for a ML application depends on several factors, such as the size and

[43] https://dreamquark-ai.github.io/tabnet/

TABLE 14. Examples of metrics for ML for Networks and Networks for ML.

Metric	Usage	Networks for ML	ML for Networks
Throughput	it measures the amount of data that can be transmitted over a network in a given time period.	implicitly impacts the convergence rate of ML models	the main objective is to optimize these metrics for network applications
Latency	it measures the time it takes for a packet to travel from the source to the destination.		
Spectral efficiency	it measures the amount of data that can be transmitted per unit of wireless bandwidth.		
Packet loss rate	it measures the proportion of packets that are lost during transmission.	also impacts the model accuracy.	
Privacy	it measures the level of data privacy that is maintained during the training/learning process	any data such as medical data or images	network data such as HTTP or IP traffic
Robustness	it measures the ability to cope with changes in the network	the ability of a model to maintain its performance when tested on new data that is different from the training data	the ability of a network to continue operating in the presence of failures or attacks.
Energy Efficiency	it relates to the power consumption of network devices	the product of power consumption per inference/training step and the amount of time for executing the task [276]	for wireless networks, it measures the amount of energy consumed per bit of data transmitted.
Fairness	for resource allocation, it measures the distribution of resources among different users or devices.	refers to correcting and eliminating bias in ML decisions. Hence, resources are data	resources are bandwidth and computation
Data Quality	it measures the quality of data that is used for training across different devices.	high-quality data may compromise privacy or suffer from high latency when collecting them. There is a trade-off between this metric and the aforementioned ones.	

complexity of the network, the nature of the data and task, and the available resources and constraints.

Examples of these constraints are computational resources and data availability. In the former, a star topology would have a central server that must have sufficient computational resources to process all the data, whereas, in a mesh topology, each node could contribute computational resources, reducing the burden on any one node. In the latter constraint, a star network topology would have the data stored in a single location, which can limit the amount of data available for training. In contrast, a mesh topology could distribute data across multiple nodes, providing a larger and more diverse data set for training. We refer to [279] for a comprehensive survey on the convergence, robustness and privacy of ML algorithms with respect to network architecture and implementation in the context of 5G networks.

In the following, we will explain advanced ML topics that have distributed implementation, exploiting both ''ML for Networks'' and ''Networks for ML'' domains.

A. CENTRALIZED ML

Centralized ML refers to training ML models on a central node of the network using data from multiple nodes and is widely applied in networked systems such as the IoT [280], [281]. Therefore, the data is first collected from various nodes in the network and then transmitted to a central server to train the ML model. Typically, the data is also preprocessed for the training, which can happen on both, the collecting nodes and the central server. In many cases, the central server has more computing resources and larger storage space than the collection nodes. Since training and inference are independent, the resulting model can be used centrally and decentrally for inference. In centralized inference, a central computing node (server) employs the model to infer from the data of various collection nodes. The collection nodes usually send their observed data to the central computing node and receive the model predictions. However, it is also common to distribute the centrally trained ML model to different nodes, which then independently infer from their local data.

Centralized ML takes advantage of monopolization through central servers (e.g., on the cloud) with powerful computing resources that can handle the processing and training of computationally-heavy models using large datasets [282]. While the increase in training speed and better resource utilization is obvious, the benefit of more accurate predictions requires a more detailed explanation. Unlike the case where each node trains its model using its local data, centralized ML training benefits from aggregating data from multiple nodes [283]. Thus, the model trains

on a larger dataset, but also the aggregated data better represents the overall data distribution, allowing the model to generalize better. For example, an ML model can extract significant information from data from different sensor types or locations. This is particularly useful in network applications such as smart cities, environment monitoring, and industrial IoT.

However, there are also several disadvantages associated with centralized ML in networked systems. First, the dependence on a central computing node for model training and inference introduces a single point of failure, and scalability issues, potentially impacting the reliability and availability of the system [284]. The centralized approach places high demands on the central server in terms of computing and network performance, making its acquisition and maintenance expensive. Secondly, the data collected by networked devices (e.g., multimedia sensors, intelligent vehicles) is transmitted in large quantities over the network, requiring high data rates. Nevertheless, transferring large amounts of data to a central node can cause network congestion and degrade real-time performance [285]. Sensitive data may need to be transmitted to the central server, potentially compromising user privacy. Recently, there are growing concerns about privacy in networked systems with data generated by networked devices, such as wearable devices or sensors, where data is often very private or sensitive [286]. This results in additional requirements for the network over which the data is transmitted, processed, and stored.

B. DISTRIBUTED ML

In various fields of application, the complexity of tasks being tackled by ML models has led to an increase in the number of model parameters. To cope with this complexity, distributed ML techniques make use of networks of interconnected computing machines to address challenges such as handling larger and distributed datasets, accommodating heightened computing resource demands, and dealing with models that surpass the memory capacity of a single machine. Here, two approaches are prevalent and usually take advantage of networking to enhance model training: 1) data-parallel and 2) model-parallel. Combinations of data- and model-parallel methods are also possible.

Data-parallel corresponds to scale-out parallelization and, therefore, increases computational capacity. During training, several machines, so-called *worker*, train instances of the ML model. These instances operate on distinct and usually non-overlapping portions of the dataset. All instances have the same model structure, number of layers, and number of neurons per layer, but the parameter values can vary. The workers periodically communicate to exchange model parameters and aggregate their updates after processing a predefined number of samples locally. Various data-parallel methods have been formulated, differing primarily in the manner of cooperation among workers during training, encompassing how workers communicate and where update aggregations occur. From this perspective, architectures can be primarily distinguished by Client-Server and Peer-to-Peer methods. The Client-Server methods use a set of decentralized workers that process model updates as *Clients* and a centralized server as *Server*. The server can be a single worker, or multiple workers organized equally or in hierarchical layers. Regardless of the server's internal structure, the server maintains the shared model state and stores all model parameters. Clients receive the current model state with its parameter set from the server and communicate their updates only to it. All communication is thus handled by the server, which can lead to a bottleneck. In contrast, Peer-to-Peer methods entail direct communication of updates among workers without the presence of a central server managing the global model state. Which workers can communicate with each other is defined in a communication topology. Here, *all-to-all* but also graph-based topologies such as *trees* and *rings* are possible. In addition to the cooperation relationship, data-parallel methods differ in whether workers transmit their updates synchronously or asynchronously and in the amount of communication overhead incurred. In production, where it is usually inferred from the model, the machines use the mutual model instance.

Model parallelism, on the other hand, splits the model and distributes it across multiple workers, allowing for model sizes larger than the memory of a single machine. Each worker trains and infers only its parts of the model, which requires less memory. Consequently, the model's entirety is upheld collectively by all workers, necessitating constant communication among them during both the training and inference phases. The data is fed to the workers that maintain the input layer of the model, and each worker forwards its computed output to the worker holding the next part of the model. In the backpropagation step during training, the workers holding the output layer first compute the updates. The updates are then propagated to the workers in reverse order and applied. A central challenge within model parallelism lies in devising an effective strategy for partitioning a given model across multiple networked machines. This partitioning determines how the model segments are distributed among workers to optimize communication and computation while maintaining overall model coherence.

Common methods for distributed ML for data-parallel and model-parallel are explained below.

C. PARAMETER SERVER

Parameter Server [287], [288] is a data-parallel Client-Server method (cf. Figure 9a). Here, multiple decentralized clients - *Worker* are connected to a centralized server - *Parameter Server*. The parameter server stores the model parameters, assigns data to workers, and aggregates the updates received from workers. Often, the parameter server is a single machine but can also be a set of equivalent or hierarchically structured machines [289]. Each worker maintains an instance of the model and individually processes parameter updates based on

its data. Typically, SGD is used for parameter optimization during a training process. The processed data can either be captured and stored on the worker machine or transmitted from the parameter server. Usually, workers access only (non-overlapping) portions of the data. The complete dataset is thus distributed across multiple workers. After processing a predefined number of data samples, the workers first propose their parameter updates to the parameter server and then receive the updated model. However, how many other workers have contributed to the updated model depends on the Parameter Server implementation. For synchronous implementations, the parameter server considers updates from all workers. The workers do not continue processing until the updated model has been broadcast. Therefore, the slowest worker impacts the time for a model update significantly. In contrast, in asynchronous implementations, the parameter server updates and broadcasts the model immediately after receiving an update from the sending worker. Here, workers proceed on different model instances. This is a problem in heterogeneous environments, with different computing resources and transmission delays. Slower workers working on outdated model instances can derange SGD's solution with their updates, causing the model to converge incorrectly. For homogeneous cluster environments, this is not the case and is often faster than synchronous systems [290]. Since synchronous and asynchronous Parameter Server implementations struggle in heterogeneous environments, time-wise and model-quality-wise, respectively, Parameter Server is typically applied in data centers.

D. FEDERATED LEARNING

Federated Learning (FL) [291] is another data-parallel Client-Server distributed ML method that enables multiple devices to collaboratively train a shared model without sharing their raw data. This approach has gained significant attention in recent years due to its ability to protect user privacy and enable learning on edge devices with limited computational resources.

In FL, multiple devices, such as smartphones, IoT devices, or edge servers, participate in the training process by locally training a model using their own data and then sending their updated model parameters to a central server. The server aggregates the updates from all devices and uses them to update the global model. Figure 9b shows an FL scenario with three connected devices and a central server. The key idea behind FL is that the global model is trained using a large amount of data from multiple devices, while each device only needs to share the model updates. This allows FL to achieve the same performance as traditional centralized learning while preserving user privacy.

One of the most widely used FL algorithms is Federated Averaging (FedAvg). FedAvg is designed to address several challenges that arise in FL, including the need to preserve data privacy, mitigate bias and inconsistency across devices, reduce communication overhead, and enable model convergence. FedAvg works by having each device train its own local model using its local data, and then the local models are aggregated to form a global model that is distributed back to the devices for further training. To address the challenge of bias and inconsistency across devices, FedAvg uses a weighted average of the local models, with the weights determined based on the amount of data each device contributes to the model. This approach ensures that each device's contribution is weighted appropriately, producing a more representative and robust global model.

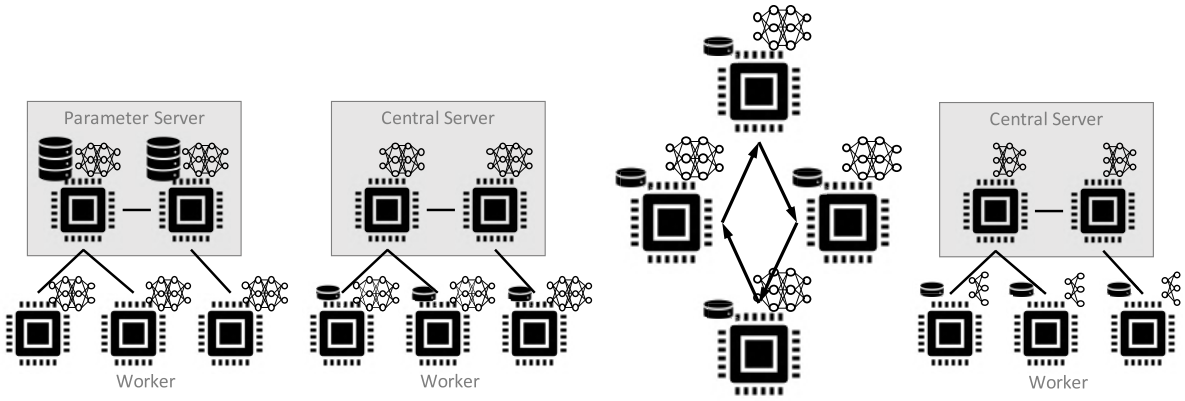
By training a model locally, FL allows devices to make predictions and decisions without the need for a constant network connection to a central server. This is particularly useful in applications such as autonomous vehicles, drones, and medical devices where data needs to be processed in real-time. Additionally, FL is also beneficial in scenarios where data is sensitive and cannot be shared, such as medical imaging or financial data. Another essential benefit of FL is its ability to handle data that is non-IID (Independent and Identically Distributed), a common characteristic of data collected from networked devices.

In traditional centralized learning, data is often assumed to be IID, which means that it has the same distribution across all devices. However, in practice, each device can have its own data distribution, which can lead to biased or suboptimal models. FL algorithms such as Federated Averaging [291], Federated Transfer Learning [292], and Federated Meta-Learning [293] are proposed to address these issues.

E. ALL-REDUCE

The All-Reduce approach [294] is a data-parallel distributed ML method for training ML models and implements the Peer-to-Peer concept. Therefore it dispenses with a central server, and instead, workers communicate directly. Which workers communicate with one another is specified by the communication topology used. Multiple communication topologies are possible for the All-Reduce approach, i.e., ring [295], butterfly [296], and trees [297]. The communication topologies affect the data rate and latency of the network differently. In some cases, the topology also restricts access to the data set.

In principle, each worker maintains an instance of the model and individually processes updates by its assigned portion of data. The data is usually distributed at the beginning of the training. After processing a predefined number of data samples, the workers communicate their local updates with all their peers. Shortly after, they receive the updates of their peers and aggregate them with their own. This step of communication and aggregation can be repeated several times. When all updates are distributed to all workers, each worker adjusts its model instance parameters according to the aggregated updates and proceeds to produce the next local updates. The repetitive and expensive communication of updates guarantees that all workers work with the same model instance [294]. Figure 9c



(a) Parameter Server (b) Federated Learning (c) Ring All-Reduce (d) Split Learning

FIGURE 9. Cooperation topologies for common distributed ML architectures.

TABLE 15. Summary of approaches of networks for ML approaches.

Method	Description	Advantages	Disadvantages
Parameter Server	a distributed system for training ML models where the model parameters are stored and updated on a centralized server and model updates are communicated from workers to the server. In contrast to Federated Learning, the data can be managed and assigned from the central server.	less complexity, efficient use of resources.	single point of failure, privacy and security concerns with centralized data storage.
Federated Learning	a method of training a ML model across multiple decentralized devices or servers while keeping data on the devices.	privacy and security of data, ability to handle large-scale decentralized data.	communication overhead, devices may have different data distributions and update rates.
All-Reduce	a data-parallel distributed ML method for training ML models and implementing a Peer-to-Peer concept.	low implementation complexity, allows parallel computation on multiple devices.	requires communication between all devices, and may be limited in terms of scalability.
Split Learning	a method of training a ML model where the model is split between two or more devices and only the output activations are transmitted between devices.	better privacy and security compared to traditional centralized learning, reduced communication overhead.	potential loss of accuracy due to approximation in forward computation, synchronization of model parameters across devices may be difficult.
Federated Split Learning	a hybrid approach that combines elements of Federated Learning and Split Learning.	achieves better privacy and security compared to traditional centralized training, reduced communication overhead compared to Federated Learning.	may be more complex to implement than other methods, devices may have different data distributions and update rates.

illustrates the communication topology of a ring All-Reduce approach.

F. SPLIT LEARNING AND INFERENCE

Split Learning (SL) [298], [299] is a model-parallel distributed ML method that decouples model training from the need for direct access to the raw data, in which a model is split into at least two sub-models. It is similar to FL, but it focuses on the case where devices have low computational power, memory constraints, or limited energy budget. In contrast to FL, where devices typically train a model locally and send the updated model parameters to a central server, in SL, the devices only forward a feature representation of their data to the central server, which performs the model updates.

In SL, the model is split into at least two parts, with one part running on the device and the other part running on the central server. Figure 9d shows the SL representation with three devices. The key idea behind SL is that the device part of the model is lightweight and can be run on devices with low computational power instead of the entire computationally demanding model. Thus, SL enables model training and inference on devices with low computing resources.

SL over networked devices is particularly useful in scenarios where devices have low computational power and high communication bandwidth. For example, in a network of smartphones, each smartphone may have a camera that captures images, but the device may not have the computational power to process the images. SL can be used to train a model that can classify images without needing to process the images on the device.

1) FEDERATED SPLIT LEARNING

Federated Split Learning (FSL) [300], [301] is a distributed algorithm that combines the ideas of computing the weighted average, a characteristic of the FL architecture, and the neural network split between the client and server of the SL architecture. It thus combines data- and model parallelism. In FSL, all clients compute in parallel and independently. They send/receive their smashed data to/from the server in parallel. The client-side sub-network synchronization, i.e., forming the global client-side network, is done by aggregating (e.g., weighted averaging) all local client-side networks on a separate server.

2) SPLIT COMPUTING

Splitting a neural network for inference tasks is usually called Split Computing (SC). It is very similar to SL, as a model is split into sub-models and then distributed on multiple devices communicating with each other. It is helpful in scenarios where sensor devices are resource-limited and can not deploy full models. Instead of offloading the sensor data, the sensor can compute a part of the model and then transmit the compressed feature representation, resulting in a smaller end-to-end latency [302].

Most works focus on a simple client-server scenario. The model is then split into a head and a tail part. The client, a sensor, gathers sensor data, feeds it into the head of the model, and then transmits the feature representation to the server. The server receives the feature representation and completes the inference process using the tail. In this client-server scenario, the main challenges are to minimize the head with regard to computation and size on the client as sensors have limited resources and to minimize the amount of communication while making sure that the model does not lose too much accuracy.

Matsubara et al. [303] provide a comprehensive survey describing many proposed methods to optimize SC. Additionally, it also contains links to code where available. With `sc2bench` [304], there is also a pip package to test and compare several SC techniques while providing a framework to start creating your own method.

VII. FURTHER READINGS

Several related survey and tutorial papers exist that cover parts of the interplay between ML and networking to a varying extent and on varying scales of granularity. Table 16 lists the most related of these papers while highlighting their ML scope, covered network applications, and whether they focus on ML for Networks (ML4N) or Networks for ML (N4ML).

Perhaps the most comprehensive survey on ML for Networks, [308] discusses ML approaches for a wide range of networking challenges and provides further references to more specialized surveys about ML approaches in certain networking domains. The work of [276] considers itself as an update to [308], covering more recent developments and discussing recent IDS datasets. Additionally, several surveys consider ML approaches for a subset of networked systems, such as vehicular networks in [316] and [317], Software-Defined Networks (SDN) in [307], mobile/wireless/ubiquitous networks in [4] and [279], edge computing [305], [306] or network traffic monitoring and analysis [314]. The work of [309] takes a unique stance and provides the joint application of recent ML and Blockchain technologies for networking problems. Other surveys focus on specific ML subdomains such as unsupervised learning [29], deep learning [272] or distributed ML [310].

The work presented in [311] and [312] specifically consider the role of FL in networking. While [311] discusses FL several applications in the domain of communications and networking, [312] focuses on mobile edge computing but also discusses how communication techniques influence FL methods. The studies [285], [313] provide an overview of various applications of ML methods through IoT systems and analyses various approaches on how ML models can be distributed and processed in the cloud-to-things continuum. The survey [284] discusses the convergence of edge computing methods and ML; specifically, it provides a comprehensive view of how networking can be utilized for cooperative processing of deep learning models on edge devices. The survey [303] provides insights into how networked devices such as smartphones and autonomous vehicles are used for collaborative training of ML model, and inference operations over the network using split computing and early exit methods.

Concerning the role of XAI in networking, the amount of survey work is limited. The work of [319] motivates the usage of XAI methods in networking challenges but only covers a single concrete problem. While there exist survey papers on XAI [320] and Explainable Reinforcement Learning (XRL) [321] in general (i.e., not limited to networking), to the best of our knowledge only [318] surveys XAI techniques in the domain of networking, namely in challenges related to wireless/6G.

VIII. CHALLENGES AND FUTURE DIRECTIONS

The adoption of ML in networks also brings forth several challenges and opens up exciting future directions for research and development for both ML for networks and Networks for ML. In this section we touch on some of these challenges, while we refer to [322] for further discussions on limitations and challenges of ML and more specifically, when we apply ML for Network [276], [323]. The following are some of the current challenges in ML for Networks:

TABLE 16. Selective surveys & tutorials on using ML for Networks (ML4N) and Networks for ML (N4ML).

Paper	Network Application	ML Scope	N4ML or ML4N
[303], [304]	edge computing: video processing, autonomous vehicles, smart homes, privacy, security, and caching	distributed training/inference and model compression: DNN, FL, Transfer Learning (TL) and shallow ML	both
[305], [306]	prediction and configuration: routing, congestion control, traffic prediction/classification, and security	shallow ML, Deep Learning (DL) and RL	ML4N
[282]	edge, fog, and cloud computing: intelligent manufacturing, security, real-time video processing, smart city	distributed ML, FL, RL, TL	N4ML
[270]	wireless and mobile computing (e.g., user localization, mobility/mobile data analysis, signal processing)	DNN: e.g., CNN, RNN and GAN; and deep RL	ML4N
[307]	traffic classification, routing optimization, resource management, security	shallow ML, DL, RL	ML4N
[4]	wireless: Virtual Reality (VR), Mobile Edge Computing (MEC), Internet of Things (IoT) and Unmanned Aerial Vehicle (UAV)	DNN: e.g., Spiking Neural Network (SNN) and RNN	both
[308]	5G: privacy and resource allocation	distributed ML, model compression	both
[301]	edge computing: video/audio processing, autonomous vehicles, smart homes, industrial IoT	Slit Computing (SC), distributed ML, Early Exit (EEoI)	N4ML
[29]	traffic classification, network optimization, and Intrusion Detection System (IDS)	unsupervised ML	ML4N
[274]	IDS, routing, network optimization, and resource allocation	supervised and unsupervised ML, RL and DNN	ML4N
[309]	IoT, vehicular networks, edge computing, wireless networks, UAV swarms	FL, RL	ML4N
[310]	edge computing: privacy, security and model compression	FL	both
[283], [311]	distributed computing: smart cities, autonomous vehicles, smart homes, industrial IoT	distributed ML, FL, EEoI	N4ML
[277]	mobile communication: privacy, security, robustness, and other exemplary applications	distributed learning: FL, multi-agent and parameter server.	both
[312]	traffic prediction/classification and security	DNN: e.g., CNN, RNN and GAN	ML4N
[313]	IoT: resource management and traffic classification	shallow ML and DL	ML4N
[314], [315]	vehicular network: security, privacy, and resource allocation	shallow ML, DL, TL and RL	ML4N
[316]	wireless/6G (e.g., signal detection, channel, and sensor network diagnostics, antenna selection)	explainable DL (e.g., symbolic representation, feature visualization, model reduction)	ML4N

- Scalability: One of the critical challenges in ML for Networks is scaling up models to handle large-scale networks with millions of nodes and edges. Most ML approaches are initially developed and tested for small-scale networks to better debug them and understand their effect on individual network components. However, making them work at scale is not always trivial because large-scale network structures might lead to computation time explosions (as has been indicated e.g. for SDN in [324]), especially for problems where global decisions are taken in a centralized manner.
- Limited data: One of the challenges in ML for Networks is the limited amount of data available for training. Collecting and labeling network data is a time-consuming and costly process, and in some cases, data may be proprietary or sensitive, making it difficult to obtain.
- Interoperability: Another challenge is the lack of interoperability of ML models. In many cases, it is difficult to understand how a model arrived at a particular decision or prediction, making it challenging to debug or troubleshoot issues.
- Heterogeneous data: Networks often contain heterogeneous data from multiple sources, such as text, images, and numerical data. Incorporating this data into ML models and designing models that can effectively handle heterogeneous data is another challenge that requires further research.
- Robustness: ML models are vulnerable to attacks and adversarial examples, especially in network environments where data may be noisy or corrupted.
- Real-time decision-making in closed-loop systems: In many Network Control Systems (NCSs) environments, decisions must be made in real-time, requiring efficient and fast ML model inference [325], [326]. Developing algorithms that can make accurate but fast decisions in real-time is a significant challenge in ML for Networks. One of the core problems is the potential for unstable system behavior caused by a mismatch between the

indented NCS sampling time and the time required for inference of an ML model. As a result, input delays affect the resulting system, which must be handled carefully [327]. Hence, there is a trade-off between large models that can handle large-scale networks (the scalability challenge) and the required time for their inference. In general, the required inference time by AI and ML models will be a non-trivial function of the resulting closed-loop system in which it is embedded. For RL, delays due to model inference can be explicitly included in the modeling, resulting in the notion of real-time MDPs and real-time RL algorithms [328]. Beyond cyber-physical closed-loop systems, model inference delay impacts user experience when prompt LLMs, IoT or VR services are run via edge computing networks [329]. In other words, in these cases, the system loop is closed via human feedback, where unstable behavior will eventually result in the performance loss.

- Energy efficiency: ML models often require significant computational resources, which can be challenging in resource-constrained network environments. As the current trend points towards ever-increasing model scales, energy efficiency might become an even more important aspect in even more situations.
- Privacy and security: Networks can contain sensitive and private data, which requires ML algorithms to be developed with strong privacy and security safeguards. ML algorithms for networks must maintain data privacy while providing accurate predictions.
- Network complexity: Computer Networks can be highly complex and dynamic, with large numbers of interconnected nodes, an interplay of various different protocols and changing operation conditions. This makes it challenging to develop accurate ML models, since formulating ML problems for complex application domains or sub-problems where suitable training data is available often requires several simplifying and/or narrowing assumptions at the start [330]. Leaving out such assumptions one by one brings ML systems closer to deployment in real-world scenarios, but often is a non-trivial task that brings unexpected challenges in every step along the way.

On the other hand, the challenges related to Networks for ML include:

- Resource constraints: ML algorithms often require significant computational resources, including processing power, memory, and storage. Moreover, the training of ML models requires large amounts of data, and transferring this data across networks can be time-consuming and resource-intensive. This can be a challenge in resource-constrained networks, such as those in IoT devices and edge computing environments, or when specialized networking hardware disallows certain compute operations. In addition, storing data in a centralized location can create a bottleneck and security issues.
- Latency: Network latency can affect the performance of ML algorithms, particularly in real-time applications where decisions must be made quickly. High latency can lead to delays in data transmission and processing, which can negatively impact the accuracy and effectiveness of the algorithm [331].
- Bandwidth: ML algorithms often require large amounts of bandwidth to transfer data, and this can be a challenge in networks with limited bandwidth. High bandwidth requirements can also lead to increased costs for network infrastructure in a real-world deployment.
- Network topology: The topology of a network can impact the performance of ML algorithms. For example, networks with high levels of congestion or interference may not be suitable for real-time applications.
- Privacy and security: ML algorithms require access to data, which can create potential privacy and security risks, increasing the risk of data breaches and cyber-attacks during transmission over the network or remote processing of user data.
- Heterogeneous resources: The computing and communication resources in devices used for processing ML algorithms over the network may vary widely, leading to unstable training processes. Furthermore, this can lead to the presence of slower devices (stragglers) that slow down the training of a global model and affect the model's efficiency.

As earlier mentioned in Section VI some of these challenges may overlap, such as privacy and security. Overall, ML for Networks and Networks for ML are rapidly growing fields with many challenges and opportunities for future research. Addressing these challenges will require collaboration between researchers from different disciplines. In the following sections, we will discuss some of the trending applications that focus on these challenges.

A. A NEW PARADIGM FOR NEXT-GENERATION WIRELESS NETWORKS

The rapid advancement of AI and ML technologies has also opened up new vistas for next-generation wireless networks like 5G Advanced and 6G. These next-generation networks essentially serve two purposes: Data transport and service delivery. They comprise various types of devices from User Equipments (UEs), base stations, switches, routers, and servers in a data center. With the integration of SDN and Network Function Virtualization (NFV), all devices can now constantly adapt to new situations, such as changing traffic patterns, better function placements, or new service demands, and incorporateAI and ML [332]. These technologies promise to revolutionize the way we design and manage wireless networks, leading to the emergence of AI-native networks and AI-native air interfaces.

On the one hand, AI-native networks are networks designed with AI integration at their core, rather than as an afterthought or add-on. Hence, AI (partially) replaces human-defined rules, models, and algorithms, which may

not be optimal or scalable for the complex and dynamic wireless scenarios, so that these networks can learn, adapt, and optimize itself autonomously and intelligently.

On the other hand, an AI-native air interface is an air interface that uses AI and ML to define and configure its physical and medium access control layer parameters, such as waveforms, constellations, pilots, coding, modulation, synchronization, channel estimation, equalization, detection, decoding, and access schemes [333].

One of the main challenges here is the complexity and heterogeneity of wireless networks. This complexity makes it difficult to collect, process, and analyze data in real-time [333]. However, this can be mitigated by using distributed AI engines, which can process data closer to the source and reduce latency. Another challenge is the lack of standardized frameworks and architectures for implementing AI in networks. To address this challenge, industry, and academia collaborate to develop standardized AI frameworks and tools that can be used across different networks [334], [335]. There are four aspects to address this challenge [336]:

1) DATA INFRASTRUCTURE

a distributed data infrastructure that can handle massive amounts of varied, distributed, and dynamic data, and enable data ingestion, processing, and exposure across layers and domains.

2) INTELLIGENCE EVERYWHERE

a comprehensive and automated management of AI models, from training to deployment to monitoring, and the ability to handle model drift, retraining, and versioning. This would take place for every network layer and on every network device.

3) ZERO TOUCH

a high degree of automation and autonomy for the management of AI and data, and the ability to express and supervise high-level goals rather than low-level actions.

4) AI AS A SERVICE

the exposure of AI and data services to external parties, such as service providers or customers, and the creation of a platform for innovation and collaboration.

For further readings on the evaluation metrics of such networks, we refer to [337]. The authors in [338] also provide a road-map with potential frameworks to build such networks.

B. DEEP NEURAL NETWORKS MODEL COMPLEXITY AND ENERGY CONSUMPTION

The increasing complexity of DNNs has direct implications on energy consumption, a critical factor in both environmental sustainability and practical deployment [339]. The complexity of DNNs is largely driven by the depth and breadth of the network architecture. As DNNs grow deeper (with more layers) and wider (with more neurons in each layer), they can capture more intricate patterns in data. This increased capacity, while maybe beneficial for model accuracy, leads to a higher number of computations during both the training and inference phases [284]. Each computation requires a certain amount of energy, and thus, as models grow more complex, their energy requirements escalate.

The energy consumption of DNNs is a multifaceted issue. Training DNNs is an energy-intensive process that requires substantial computational resources [340]. This phase often necessitates the use of high-performance GPUs or even clusters of GPUs, which are power-hungry devices [341]. The electricity consumption during this phase is considerable, contributing to the overall energy footprint of developing DNNs. The inference phase, where DNNs make predictions on new data, also demands a considerable amount of energy [342]. This phase is critical in real-world applications where continuous or on-demand operation of DNNs is required, such as in autonomous systems or real-time analysis applications.

The substantial energy consumption of DNNs poses a significant challenge for environmental sustainability. As these networks become more prevalent across various sectors, the need for energy-efficient neural network architectures and training methods becomes increasingly important [343]. In energy-constrained environments (e.g., with battery-operated devices) the energy demands of DNNs are a crucial consideration. This has led to a focus on balancing model complexity with energy efficiency, driving innovation in optimization techniques, and the development of specialized hardware to run these models more efficiently [344]. Moreover, different models and benchmarks are used to estimate and plan the energy consumption of DNNs [345], [346], [347].

C. TINY MACHINE LEARNING

Tiny Machine Learning (TinyML) is an emerging field that combines ML with ultra-low power computing, typically found in microcontrollers and small IoT devices [348]. Its goal is to deploy efficient ML models that can operate in environments with limited memory, processing power, and energy. This is particularly relevant for applications where traditional ML models would be impractical due to their size and energy requirements.

The primary motivation for TinyML is the need for localized data processing, especially in situations where privacy, speed, and power efficiency are critical, rather than transmitting it to a centralized server or cloud [349]. This can be applied for many applications, spanning from smart home devices and wearable technology to healthcare monitoring and environmental sensors [285].

The core implementation of TinyML relies on ML model quantization, which reduces its numerical precision and size. Hence, implementing TinyML in environments with limited resources presents several ongoing challenges – The

low computational capabilities and storage capacities of smaller devices restrict the complexity of the models that can be deployed [350]. This constraint can adversely affect the efficacy and precision of TinyML-based applications. To address this, some research suggests the integration of cooperative ML (Section VI) and TinyML approaches [342], [351]. This strategy would enable devices with constrained resources to work collaboratively on ML tasks. Moreover, progress in hardware development, particularly in creating more efficient microcontrollers and sensors, is expected to broaden the range of possible applications for TinyML. For a recent survey of tinyML applications and techniques, we refer to [352].

IX. CONCLUSION

The aim of this paper is to provide interested but inexperienced readers an an inspiring and practical jumpstart for research in the intersection of ML and computer networking. This encompasses not only the creation of novel ML-powered solutions for covered networking scenarios but also leveraging established networking technology to enhance existing ML approaches.

Compared to the aforementioned surveys and tutorials (Section VII), we are the first to provide a comprehensive bidirectional overview of ML and XAI techniques across different networking fields, and vice versa.[44] Furthermore, in addition to an overview of the current state of the art, our work provides practical guidance for aspiring researchers to shortcut their way into meaningful research:

- Many of the mentioned related papers do not consider datasets and/or starting points to reproduce the results or even to just start experimenting. In contrast, we refer to publicly available datasets as well as methods and tools to generate synthetic datasets (Section IV) and design ML models suitable for the respective task.
- We categorize existing approaches as ML serving networks (ML4N) and networks serving ML (N4ML) based on the used metrics, which helps to identify research gaps and possible future directions of research.

We introduced the most popular ML techniques, model types, and tools as well as several practical aspects to consider when practicing ML such as obtaining high-quality data for the learning algorithm, or the incorporation of inductive biases (more specifically for networking data and network topologies) into ML models in order to reduce resource requirements. Secondly, we introduced the most common computer networking problem domains and pointed to existing tools and datasets to accelerate and facilitate ML research on networking problems.

Thirdly, we introduced how XAI methods can improve the transparency of ML models' decisions and thus push their acceptance in the computer networks research domain and their suitability in productive environments. We also elaborated on how networking techniques can boost the performance of existing ML setups and workflows, e.g. through several approaches for distributed learning.

Lastly, we provided a large number of pointers for further reading, such as surveys on more specific ML/networking domains, example research works for some of the problems introduced in this paper or links to many of the mentioned datasets or tools.

Despite our comprehensive coverage of established tools, approaches, and recent breakthroughs, it's important to acknowledge the dynamic nature of ML research. The field is characterized by the emergence of new algorithms, the potential availability of additional tools and features in the future, and the hopeful prospect of more open-sourced datasets. While this evolution is happening at an unprecedented pace, this paper still serves as a valuable starting point for researchers and newcomers alike and provides a timely and relevant contribution to the intersection of the fields of ML and computer networking.

[44] We do not aim for a comprehensive review of state-of-the-art research in ML or its sub-disciplines, as there are numerous survey and tutorial resources that provide an excellent ML-focused overview. Rather, we view ML techniques solely in relation to networking, either as facilitators (ML for Networks) or beneficiaries (Networks for ML).

ACKNOWLEDGMENT

The authors alone are responsible for the content. This work is a result of a cooperation and continuous knowledge exchange between participants of the MaLeNe Workshop 2022.

REFERENCES

[1] J. M. Stokes, K. Yang, K. Swanson, W. Jin, A. Cubillos-Ruiz, N. M. Donghia, C. R. MacNair, S. French, L. A. Carfrae, Z. Bloom-Ackermann, V. M. Tran, A. Chiappino-Pepe, A. H. Badran, I. W. Andrews, E. J. Chory, G. M. Church, E. D. Brown, T. S. Jaakkola, R. Barzilay, and J. J. Collins, "A deep learning approach to antibiotic discovery," *Cell*, vol. 180, no. 4, pp. 688–702, Feb. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0092867420301021

[2] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2022, pp. 10674–10685.

[3] A. Davies, P. Veličković, L. Buesing, S. Blackwell, D. Zheng, N. Tomašev, R. Tanburn, P. Battaglia, C. Blundell, A. Juhász, M. Lackenby, G. Williamson, D. Hassabis, and P. Kohli, "Advancing mathematics by guiding human intuition with AI," *Nature*, vol. 600, no. 7887, pp. 70–74, Dec. 2021. [Online]. Available: https://www.nature.com/articles/s41586-021-04086-x

[4] M. Chen, U. Challita, W. Saad, C. Yin, and M. Debbah, "Artificial neural networks-based machine learning for wireless networks: A tutorial," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3039–3071, 4th Quart., 2019.

[5] S. J. Russell, *Artificial Intelligence a Modern Approach*. London, U.K.: Pearson, 2010.

[6] M. F. A. Fauzi, R. Nordin, N. F. Abdullah, and H. A. H. Alobaidy, "Mobile network coverage prediction based on supervised machine learning algorithms," *IEEE Access*, vol. 10, pp. 55782–55793, 2022.

[7] C. Ioannou and V. Vassiliou, "Classifying security attacks in IoT networks using supervised learning," in *Proc. 15th Int. Conf. Distrib. Comput. Sensor Syst. (DCOSS)*, May 2019, pp. 652–658.

[8] W. Hu, Y. Liao, and R. Vemuri, "Robust anomaly detection using support vector machines," *Proc. Int. Conf. Mach. Learn.*, Jun. 2003, pp. 282–289.

[9] B. Mohammed, I. Awan, H. Ugail, and M. Younas, "Failure prediction using machine learning in a virtualised HPC system and application," *Cluster Comput.*, vol. 22, no. 2, pp. 471–485, Jun. 2019.

[10] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intell. Syst. Appl.*, vol. 13, no. 4, pp. 18–28, Aug. 1998.

[11] S. B. Kotsiantis, "Decision trees: A recent overview," *Artif. Intell. Rev.*, vol. 39, no. 4, pp. 261–283, Apr. 2013.
[12] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, pp. 5–32, Oct. 2001.
[13] G. Shakhnarovich, T. Darrell, and P. Indyk, "Nearest-neighbor methods in learning and vision," *IEEE Trans. Neural Netw.*, vol. 19, no. 2, p. 377, Feb. 2008.
[14] M. Nasri and M. Hamdi, "LTE QoS parameters prediction using multivariate linear regression algorithm," in *Proc. 22nd Conf. Innov. Clouds, Internet Netw. Workshops (ICIN)*, Feb. 2019, pp. 145–150.
[15] A. Y. Nikravesh, S. A. Ajila, C.-H. Lung, and W. Ding, "Mobile network traffic prediction using MLP, MLPWD, and SVM," in *Proc. IEEE Int. Congr. Big Data (BigData Congr.)*, Jun. 2016, pp. 402–409.
[16] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statist. Comput.*, vol. 14, no. 3, pp. 199–222, Aug. 2004.
[17] C.-Y. Hsu, P.-Y. Chen, S. Lu, S. Liu, and C.-M. Yu, "Adversarial examples can be effective data augmentation for unsupervised machine learning," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 6926–6934.
[18] D. Kim and J. Choi, "Unsupervised representation learning for binary networks by joint classifier learning," 2021, *arXiv:2110.08851*.
[19] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "DBSCAN revisited, revisited: Why and how you should (Still) use DBSCAN," *ACM Trans. Database Syst.*, vol. 42, no. 3, pp. 1–21, Sep. 2017.
[20] J. Li, H. Izakian, W. Pedrycz, and I. Jamal, "Clustering-based anomaly detection in multivariate time series data," *Appl. Soft Comput.*, vol. 100, Mar. 2021, Art. no. 106919.
[21] I. Ullah and H. Y. Youn, "Task classification and scheduling based on K-means clustering for edge computing," *Wireless Pers. Commun.*, vol. 113, no. 4, pp. 2611–2624, Aug. 2020.
[22] Z. Fan and R. Liu, "Investigation of machine learning based network traffic classification," in *Proc. Int. Symp. Wireless Commun. Syst. (ISWCS)*, Aug. 2017, pp. 1–6.
[23] R. Bellman, "Dynamic programming," *Science*, vol. 153, nos. 37–31, pp. 34–37, 1966.
[24] H. Abdi and L. J. Williams, "Principal component analysis," *WIREs Comput. Statistic*, vol. 2, no. 4, pp. 433–459, Jul./Aug. 2010.
[25] C. Fefferman, S. Mitter, and H. Narayanan, "Testing the manifold hypothesis," *J. Amer. Math. Soc.*, vol. 29, no. 4, pp. 983–1049, Feb. 2016.
[26] U. Narayanan, A. Unnikrishnan, V. Paul, and S. Joseph, "A survey on various supervised classification algorithms," in *Proc. Int. Conf. Energy, Commun., Data Anal. Soft Comput. (ICECDS)*, Aug. 2017, pp. 2118–2124.
[27] J. E. van Engelen and H. H. Hoos, "A survey on semi-supervised learning," *Mach. Learn.*, vol. 109, no. 2, pp. 373–440, Feb. 2020, doi: 10.1007/s10994-019-05855-6.
[28] M. A. Alsheikh, S. Lin, D. Niyato, and H.-P. Tan, "Machine learning in wireless sensor networks: Algorithms, strategies, and applications," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 4, pp. 1996–2018, 4th Quart., 2014.
[29] M. Usama, J. Qadir, A. Raza, H. Arif, K. A. Yau, Y. Elkhatib, A. Hussain, and A. Al-Fuqaha, "Unsupervised machine learning for networking: Techniques, applications and research challenges," *IEEE Access*, vol. 7, pp. 65579–65615, 2019.
[30] Z. Ghahramani, "Probabilistic machine learning and artificial intelligence," *Nature*, vol. 521, no. 7553, pp. 452–459, May 2015. [Online]. Available: https://www.nature.com/articles/nature14541
[31] K. P. Murphy, *Probabilistic Machine Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2022. [Online]. Available: https://probml.github.io/pml-book/book1.html
[32] X. Liu, F. Zhang, Z. Hou, L. Mian, Z. Wang, J. Zhang, and J. Tang, "Self-supervised learning: Generative or contrastive," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 1, pp. 857–876, Jan. 2023.
[33] F. Ebert, C. Finn, A. X. Lee, and S. Levine, "Self-supervised visual planning with temporal skip connections," in *Proc. Conf. Robot Learn.*, 2017, pp. 1–13.
[34] S. Meyn, *Control Systems and Reinforcement Learning*. Cambridge, U.K.: Cambridge Univ. Press, 2022.
[35] Y. Xu, G. Gui, H. Gacanin, and F. Adachi, "A survey on resource allocation for 5G heterogeneous networks: Current research, future trends, and challenges," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 2, pp. 668–695, 2nd Quart., 2021.
[36] M. M. Sadeeq, N. M. Abdulkareem, S. R. M. Zeebaree, D. M. Ahmed, A. S. Sami, and R. R. Zebari, "IoT and cloud computing issues, challenges and opportunities: A review," *Qubahan Academic J.*, vol. 1, no. 2, pp. 1–7, Mar. 2021.
[37] P. Kumar and R. Kumar, "Issues and challenges of load balancing techniques in cloud computing: A survey," *ACM Comput. Surveys*, vol. 51, no. 6, pp. 1–35, Nov. 2019.
[38] A. Alwarafy, M. Abdallah, B. S. Ciftler, A. Al-Fuqaha, and M. Hamdi, "Deep reinforcement learning for radio resource allocation and management in next generation heterogeneous wireless networks: A survey," 2021, *arXiv:2106.00574*.
[39] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[40] D. Bertsekas, *Dynamic Programming and Optimal Control*, vol. 2. Nashua, NH, USA: Athena Scientific, 2012.
[41] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, vol. 5. Belmont, MA, USA: Athena Scientific, 1996.
[42] T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker, "Model-based reinforcement learning: A survey," *Found. Trends Mach. Learn.*, vol. 16, no. 1, pp. 1–118, 2023.
[43] Y.-P. Hsu, E. Modiano, and L. Duan, "Age of information: Design and analysis of optimal scheduling algorithms," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jun. 2017, pp. 561–565.
[44] Q. Sykora, M. Ren, and R. Urtasun, "Multi-agent routing value iteration network," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 9300–9310.
[45] S. S. Mwanje, L. C. Schmelz, and A. Mitschele-Thiel, "Cognitive cellular networks: A Q-learning framework for self-organizing networks," *IEEE Trans. Netw. Service Manage.*, vol. 13, no. 1, pp. 85–98, Mar. 2016.
[46] Y. Kim, S. Kim, and H. Lim, "Reinforcement learning based resource management for network slicing," *Appl. Sci.*, vol. 9, no. 11, p. 2361, Jun. 2019.
[47] H. Afifi and H. Karl, "Reinforcement learning for virtual network embedding in wireless sensor networks," in *Proc. 16th Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob)*, Oct. 2020, pp. 123–128.
[48] A. Geramifard, "A tutorial on linear function approximators for dynamic programming and reinforcement learning," *Found. Trends Mach. Learn.*, vol. 6, no. 4, pp. 375–451, 2013.
[49] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 12, 1999, pp. 1–12.
[50] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," in *Reinforcement Learning*. Boston, MA, USA: Springer, 1992, pp. 5–32.
[51] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Trans. Syst. Man, Cybern. C, Appl. Rev.*, vol. 42, no. 6, pp. 1291–1307, Nov. 2012.
[52] V. Mnih, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.
[53] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. 15th ACM Workshop Hot Topics Netw.*, Nov. 2016, pp. 50–56.
[54] C. Zhong, Z. Lu, M. C. Gursoy, and S. Velipasalar, "A deep actor-critic reinforcement learning framework for dynamic multichannel access," *IEEE Trans. Cognit. Commun. Netw.*, vol. 5, no. 4, pp. 1125–1139, Dec. 2019.
[55] S. Tuli, S. Ilager, K. Ramamohanarao, and R. Buyya, "Dynamic scheduling for stochastic edge-cloud computing environments using A3C learning and residual recurrent neural networks," *IEEE Trans. Mobile Comput.*, vol. 21, no. 3, pp. 940–954, Mar. 2022.
[56] M. Chen, T. Wang, K. Ota, M. Dong, M. Zhao, and A. Liu, "Intelligent resource allocation management for vehicles network: An A3C learning approach," *Comput. Commun.*, vol. 151, pp. 485–494, Feb. 2020.
[57] S. Still and D. Precup, "An information-theoretic approach to curiosity-driven reinforcement learning," *Theory Biosciences*, vol. 131, no. 3, pp. 139–148, Sep. 2012.
[58] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," 2018, *arXiv:1810.12894*.
[59] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Proc. Mach. Learn.*, Jan. 1994, pp. 157–163.
[60] T. Gabel, "Multi-agent reinforcement learning approaches for distributed job shop scheduling problems," Ph.D. dissertation, Dept. Math. Comput. Sci., Osnabrück Univ., Osnabrück, Germany, 2009.
[61] L. Canese, G. C. Cardarilli, L. Di Nunzio, R. Fazzolari, D. Giardino, M. Re, and S. Spanò, "Multi-agent reinforcement learning: A review of challenges and applications," *Appl. Sci.*, vol. 11, no. 11, p. 4948, May 2021.

[62] T. Li, K. Zhu, N. C. Luong, D. Niyato, Q. Wu, Y. Zhang, and B. Chen, "Applications of multi-agent reinforcement learning in future internet: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 2, pp. 1240–1279, 2nd Quart., 2022.

[63] E. Altman, *Constrained Markov Decision Processes*, vol. 7. Boca Raton, FL, USA: CRC Press, 1999.

[64] S. Gu, L. Yang, Y. Du, G. Chen, F. Walter, J. Wang, Y. Yang, and A. Knoll, "A review of safe reinforcement learning: Methods, theory and applications," 2022, *arXiv:2205.10330*.

[65] A. Avranas, M. Kountouris, and P. Ciblat, "Deep reinforcement learning for resource constrained multiclass scheduling in wireless networks," 2020, *arXiv:2011.13634*.

[66] S. Khairy, P. Balaprakash, L. X. Cai, and Y. Cheng, "Constrained deep reinforcement learning for energy sustainable multi-UAV based random access IoT networks with NOMA," 2020, *arXiv:2002.00073*.

[67] C. Sun, C. She, and C. Yang, "Unsupervised deep learning for optimizing wireless systems with instantaneous and statistic constraints," 2020, *arXiv:2006.01641*.

[68] *Constrained Unsupervised Learning for Wireless Network Optimization*. Cambridge, U.K.: Cambridge Univ. Press, 2022, pp. 182–211.

[69] D. Wu, L. Deng, Z. Liu, Y. Zhang, and Y. S. Han, "Reinforcement learning random access for delay-constrained heterogeneous wireless networks: A two-user case," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2021, pp. 1–7.

[70] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[71] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.

[72] T. B. Brown, "Language models are few-shot learners," in *Proc. NIPS*, 2020, pp. 1877–1901. https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[73] W. Mcculloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biol.*, vol. 52, nos. 1–2, pp. 99–115, 1990.

[74] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *Towards Data Sci.*, vol. 6, no. 12, pp. 310–316, 2017.

[75] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, Jan. 1989. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208

[76] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural Networks for Perception*. Amsterdam, The Netherlands: Elsevier, 1992, pp. 65–93.

[77] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.

[78] G. Lan, *First-order and Stochastic Optimization Methods for Machine Learning*, vol. 1. Cham, Switzerland: Springer, 2020.

[79] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, "Geometric deep learning: Grids, groups, graphs, geodesics, and gauges," 2021, *arXiv:2104.13478*.

[80] R. Eldan and O. Shamir, "The power of depth for feedforward neural networks," in *Proc. Conf. Learn. Theory*, 2016, pp. 907–940.

[81] T. Gruber, S. Cammerer, J. Hoydis, and S. T. Brink, "On deep learning-based channel decoding," in *Proc. 51st Annu. Conf. Inf. Sci. Syst. (CISS)*, Mar. 2017, pp. 1–6.

[82] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, and N. D. Sidiropoulos, "Learning to optimize: Training deep neural networks for wireless resource management," in *Proc. IEEE 18th Int. Workshop Signal Process. Adv. Wireless Commun. (SPAWC)*, Jul. 2017, pp. 1–6.

[83] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *Proc. Int. Conf. Wireless Netw. Mobile Commun. (WINCOM)*, Oct. 2016, pp. 258–263.

[84] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[85] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1724–1734. [Online]. Available: http://aclweb.org/anthology/D14-1179

[86] P. Veličković, "Everything is connected: Graph neural networks," 2023, *arXiv:2301.08210*.

[87] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, Eds., vol. 27. Red Hook, NY, USA: Curran Associates, 2014, pp. 2672–2680. [Online]. Available: https://proceedings.neurips.cc/paperfiles/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf

[88] C. Han, H. Hayashi, L. Rundo, R. Araki, W. Shimoda, S. Muramatsu, Y. Furukawa, G. Mauri, and H. Nakayama, "GAN-based synthetic brain MR image generation," in *Proc. IEEE 15th Int. Symp. Biomed. Imag. (ISBI)*, Apr. 2018, pp. 734–738.

[89] Y. Chen, Y. Pan, T. Yao, X. Tian, and T. Mei, "Mocycle-GAN: Unpaired video-to-video translation," in *Proc. 27th ACM Int. Conf. Multimedia*, Oct. 2019, pp. 647–655.

[90] J. Kong, J. Kim, and J. Bae, "HiFi-GAN: Generative adversarial networks for efficient and high fidelity speech synthesis," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 17022–17033.

[91] A. Cheng, "PAC-GAN: Packet generation of network traffic using generative adversarial networks," in *Proc. IEEE 10th Annu. Inf. Technol., Electron. Mobile Commun. Conf. (IEMCON)*, Oct. 2019, pp. 0728–0734.

[92] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014, *arXiv:1409.0473*.

[93] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–14.

[94] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

[95] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16×16 words: Transformers for image recognition at scale," 2020, *arXiv:2010.11929*.

[96] C. Joshi, "Transformers are graph neural networks," in *The Gradient*, vol. 12, 2020. Accessed: Apr. 12, 2024. [Online]. Available: https://thegradient.pub/transformers-are-graph-neural-networks/

[97] D. K. Kholgh and P. Kostakos, "PAC-GPT: A novel approach to generating synthetic network traffic with GPT-3," *IEEE Access*, vol. 11, pp. 114936–114951, 2023.

[98] N. Ziems, G. Liu, J. Flanagan, and M. Jiang, "Explaining tree model decisions in natural language for network intrusion detection," 2023, *arXiv:2310.19658*.

[99] T. Ali and P. Kostakos, "HuntGPT: Integrating machine learning-based anomaly detection and explainable AI with large language models (LLMs)," 2023, *arXiv:2309.16021*.

[100] S. K. Mani, Y. Zhou, K. Hsieh, S. Segarra, T. Eberl, E. Azulai, I. Frizler, R. Chandra, and S. Kandula, "Enhancing network management using code generated by large language models," in *Proc. 22nd ACM Workshop Hot Topics Netw.*, Nov. 2023, pp. 196–204.

[101] Y. Huang, H. Du, X. Zhang, D. Niyato, J. Kang, Z. Xiong, S. Wang, and T. Huang, "Large language models for networking: Applications, enabling techniques, and challenges," 2023, *arXiv:2311.17474*.

[102] J. Sun, Q. V. Liao, M. Müller, M. Agarwal, S. Houde, K. Talamadupula, and J. D. Weisz, "Investigating explainability of generative AI for code through scenario-based design," in *Proc. 27th Int. Conf. Intell. User Interface*. New York, NY, USA: Association for Computing Machinery, Mar. 2022, pp. 212–228, doi: 10.1145/3490099.3511119.

[103] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, "Retrieval-augmented generation for large language models: A survey," 2023, *arXiv:2312.10997*.

[104] Cisco. (2023). *Cisco Unveils Next-Gen Solutions That Empower Security and Productivity With Generative AI*. [Online]. Available: https://newsroom.cisco.com/c/r/newsroom/en/us/a/y2023/m06/cisco-unveils-next-gen-solutions-that-empower-security-and-productivity-with-generative-ai.html

[105] Juniper. (2023). *AI for IT Operations (AIOps)*. [Online]. Available: https://www.juniper.net/us/en/solutions/artificial-intelligence-for-it-operations-aiops.html

[106] O. Santos. (2023). *Securing AI: Navigating the Complex Landscape of Models, Fine-Tuning, and Rag*. [Online]. Available: https://blogs.cisco.com/security/securing-ai-navigating-the-complex-landscape-of-models-fine-tuning-and-rag

[107] I. Cisco Systems. (2023). *Cisco AI Assistant Cisco*. [Online]. Available: https://www.cisco.com/site/us/en/solutions/artificial-intelligence/ai-assistant/index.html

[108] S. Thrun and A. Schwartz, "Issues in using function approximation for reinforcement learning," in *Proc. 4th Connectionist Models Summer School*, vol. 255, 1993, p. 263.

[109] V. Mnih, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[110] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.

[111] A. Ramaswamy, S. Bhatnagar, and N. Saxena, "A framework for provably stable and consistent training of deep feedforward networks," 2023, *arXiv:2305.12125*.

[112] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," *PLoS ONE*, vol. 12, no. 4, Apr. 2017, Art. no. e0172395.

[113] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–21.

[114] A. Redder, A. Ramaswamy, and H. Karl, "3DPG: Distributed deep deterministic policy gradient algorithms for networked multi-agent systems," 2022, *arXiv:2201.00570*.

[115] C. Qiu, H. Yao, F. R. Yu, F. Xu, and C. Zhao, "Deep Q-learning aided networking, caching, and computing resources allocation in software-defined satellite-terrestrial networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 6, pp. 5871–5883, Jun. 2019.

[116] S. Schneider, R. Khalili, A. Manzoor, H. Qarawlus, R. Schellenberg, H. Karl, and A. Hecker, "Self-learning multi-objective service coordination using deep reinforcement learning," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 3, pp. 3829–3842, Sep. 2021.

[117] A. Redder, A. Ramaswamy, and D. E. Quevedo, "Deep reinforcement learning for scheduling in large-scale networked control systems," *IFAC-PapersOnLine*, vol. 52, no. 20, pp. 333–338, 2019.

[118] H. Afifi, A. Ramaswamy, and H. Karl, "Reinforcement learning for autonomous vehicle movements in wireless sensor networks," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2021, pp. 1–6.

[119] B. Jang, M. Kim, G. Harerimana, and J. W. Kim, "Q-learning algorithms: A comprehensive classification and applications," *IEEE Access*, vol. 7, pp. 133653–133667, 2019.

[120] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3133–3174, 4th Quart., 2019.

[121] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," 2012, *arXiv:1207.0580*.

[122] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, "Commute path bandwidth traces from 3G networks: Analysis and applications," in *Proc. 4th ACM Multimedia Syst. Conf.*, Feb. 2013, pp. 114–118.

[123] X. Zuo, J. Yang, M. Wang, and Y. Cui, "Adaptive bitrate with user-level QoE preference for video streaming," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2022, pp. 1279–1288.

[124] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. De Turck, "HTTP/2-based adaptive streaming of HEVC video over 4G/LTE networks," *IEEE Commun. Lett.*, vol. 20, no. 11, pp. 2177–2180, Nov. 2016.

[125] L. Zhang, Y. Zhang, X. Wu, F. Wang, L. Cui, Z. Wang, and J. Liu, "Batch adaptative streaming for video analytics," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2022, pp. 2158–2167.

[126] A. Alhilal, T. Braud, B. Han, and P. Hui, "Nebula: Reliable low-latency video transmission for mobile cloud gaming," in *Proc. ACM Web Conf.*, Apr. 2022, pp. 3407–3417.

[127] D. Raca, J. J. Quinlan, A. H. Zahran, and C. J. Sreenan, "Beyond throughput: A 4G LTE dataset with channel and context metrics," in *Proc. 9th ACM Multimedia Syst. Conf.*, Jun. 2018, pp. 460–465.

[128] S. Farthofer, M. Herlich, C. Maier, S. Pochaba, J. Lackner, and P. Dorfinger, "An open mobile communications drive test data set and its use for machine learning," *IEEE Open J. Commun. Soc.*, vol. 3, pp. 1688–1701, 2022.

[129] J. Wu, L. Wang, Q. Pei, X. Cui, F. Liu, and T. Yang, "HiTDL: High-throughput deep learning inference at the hybrid mobile edge," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 12, pp. 4499–4514, Dec. 2022.

[130] D. Raca, D. Leahy, C. J. Sreenan, and J. J. Quinlan, "Beyond throughput, the next generation: A 5G dataset with channel and context metrics," in *Proc. 11th ACM Multimedia Syst. Conf.*, May 2020, pp. 303–308.

[131] *Geant/Abilene Network Topology Data and Traffic Traces*, 3rd Party, Ocala, FL, USA, 2020.

[132] *GENI*. Accessed: Apr. 12, 2024. [Online]. Available: https://www.geni.net/

[133] (Nov. 2020). *Caida Data Completed Datasets*. [Online]. Available: https://www.caida.org/catalog/datasets/completed-datasets/

[134] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring ISP topologies with rocketfuel," *ACM Trans. Netw.*, vol. 12, no. 1, pp. 2–16, 2004.

[135] (2021). *The Internet Topology Zoo*. [Online]. Available: http://www.topology-zoo.org/dataset.html

[136] M. Roughan, "A case study of the accuracy of SNMP measurements," *J. Electr. Comput. Eng.*, vol. 2010, pp. 1–7, May 2010, doi: 10.1155/2010/812979.

[137] J. Kua, G. Armitage, and P. Branch, "A survey of rate adaptation techniques for dynamic adaptive streaming over HTTP," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1842–1866, 3rd Quart., 2017.

[138] G. Zhou, R. Wu, M. Hu, Y. Zhou, T. Z. J. Fu, and D. Wu, "Vibra: Neural adaptive streaming of VBR-encoded videos," in *Proc. 31st ACM Workshop Netw. Operating Syst. Support Digit. Audio Video*, Jul. 2021, pp. 1–8.

[139] Y. Yuan, W. Wang, Y. Wang, S. S. Adhatarao, B. Ren, K. Zheng, and X. Fu, "VSiM: Improving QoE fairness for video streaming in mobile environments," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2022, pp. 1309–1318.

[140] S. Lederer, C. Müller, and C. Timmerer, "Dynamic adaptive streaming over HTTP dataset," in *Proc. 3rd Multimedia Syst. Conf.*, Feb. 2012, pp. 89–94.

[141] S. Lederer, C. Mueller, C. Timmerer, C. Concolato, J. Le Feuvre, and K. Fliegel, "Distributed dash dataset," in *Proc. 4th ACM Multimedia Syst. Conf.*, 2013, pp. 131–135.

[142] J. Le Feuvre, J.-M. Thiesse, M. Parmentier, M. Raulet, and C. Daguet, "Ultra high definition HEVC DASH data set," in *Proc. 5th ACM Multimedia Syst. Conf.*, Mar. 2014, pp. 7–12.

[143] A. Zabrovskiy, C. Feldmann, and C. Timmerer, "Multi-codec DASH dataset," in *Proc. 9th ACM Multimedia Syst. Conf.*, Jun. 2018, pp. 438–443.

[144] A. Chandramohan, M. Poel, B. Meijerink, and G. Heijenk, "Machine learning for cooperative driving in a multi-lane highway environment," in *Proc. Wireless Days (WD)*, Apr. 2019, pp. 1–4.

[145] L. N. Alegre, T. Ziemke, and A. L. C. Bazzan, "Using reinforcement learning to control traffic signals in a real-world scenario: An approach based on linear function approximation," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 7, pp. 9126–9135, Jul. 2022.

[146] C. Liu, Y. Zhang, W. Chen, F. Wang, H. Li, and Y.-D. Shen, "Adaptive matching strategy for multi-target multi-camera tracking," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2022, pp. 2934–2938.

[147] M. Maciejewski, "A comparison of microscopic traffic flow simulation systems for an urban area," *Transp. Problems*, vol. 5, no. 4, pp. 29–40, 2010.

[148] F. K. Karnadi, Z. H. Mo, and K.-C. Lan, "Rapid generation of realistic mobility models for VANET," in *Proc. IEEE Wireless Commun. Netw. Conf.*, Mar. 2007, pp. 2506–2511.

[149] M. Tsao, D. Milojevic, C. Ruch, M. Salazar, E. Frazzoli, and M. Pavone, "Model predictive control of ride-sharing autonomous mobility-on-demand systems," in *Proc. Int. Conf. Robot. Autom. (ICRA)*, May 2019, pp. 6665–6671.

[150] C. M. Moyano, J. F. Ortega, and D. E. Mogrovejo, "Efficiency analysis during calibration of traffic microsimulation models in conflicting intersections near Universidad del Azuay, using Aimsun 8.1," in *Proc. MOVICI-MOYCOT Joint Conf. Urban Mobility Smart City*, Apr. 2018, pp. 1–6.

[151] L. Yang and W. Lan, "On secondary development of PTV-VISSIM for traffic optimization," in *Proc. 13th Int. Conf. Comput. Sci. Educ. (ICCSE)*, Aug. 2018, pp. 1–5.

[152] L. Lu, T. Yun, L. Li, Y. Su, and D. Yao, "A comparison of phase transitions produced by PARAMICS, TransModeler, and VISSIM," *IEEE Intell. Transp. Syst. Mag.*, vol. 2, no. 3, pp. 19–24, Fall. 2010.

[153] Z. Tang, M. Naphade, M.-Y. Liu, X. Yang, S. Birchfield, S. Wang, R. Kumar, D. Anastasiu, and J.-N. Hwang, "CityFlow: A city-scale benchmark for multi-target multi-camera vehicle tracking and re-identification," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 8789–8798.

[154] Z. Wang, B. Li, and B. Liang, "Quick: Quality-of-service improvement with cooperative relaying and network coding," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2010, pp. 1–5.

[155] T. Mangla, E. Halepovic, M. Ammar, and E. Zegura, "EMIMIC: Estimating HTTP-based video QoE metrics from encrypted network traffic," in *Proc. Netw. Traffic Meas. Anal. Conf. (TMA)*, Jun. 2018, pp. 1–8.

[156] C. Gutterman, K. Guo, S. Arora, X. Wang, L. Wu, E. Katz-Bassett, and G. Zussman, "Requet: Real-time QoE detection for encrypted Youtube traffic," in *Proc. 10th ACM Multimedia Syst. Conf.*, Jun. 2019, pp. 48–59.

[157] M. Seufert, P. Casas, N. Wehner, L. Gang, and K. Li, "Stream-based machine learning for real-time QoE analysis of encrypted video streaming traffic," in *Proc. 22nd Conf. Innov. Clouds, Internet Netw. Workshops (ICIN)*, Feb. 2019, pp. 76–81.

[158] N. Wehner, M. Ring, J. Schüler, A. Hotho, T. Hoßfeld, and M. Seufert, "On learning hierarchical embeddings from encrypted network traffic," in *Proc. NOMS IEEE/IFIP Netw. Oper. Manage. Symp.*, Apr. 2022, pp. 1–7.

[159] K. Dietz, M. Mühlhauser, M. Seufert, N. Gray, T. Hoßfeld, and D. Herrmann, "Browser fingerprinting: How to protect machine learning models and data with differential privacy?" *Electron. Commun. EASST*, vol. 80, pp. 1–7, Sep. 2021.

[160] N. Wehner, M. Seufert, J. Schüler, P. Casas, and T. Hoßfeld, "How are your apps doing? QoE inference and analysis in mobile devices," in *Proc. 17th Int. Conf. Netw. Service Manage. (CNSM)*, Oct. 2021, pp. 49–55.

[161] A. Azab, M. Khasawneh, S. Alrabaee, K.-K.-R. Choo, and M. Sarsour, "Network traffic classification: Techniques, datasets, and challenges," *Digit. Commun. Netw.*, Sep. 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352864822001845

[162] D. Shamsimukhametov, M. Liubogoshchev, E. Khorov, and I. Akyildiz, "Youtube Netflix web dataset for encrypted traffic classification," in *Proc. Int. Conf. Eng. Telecommun.*, 2021, pp. 1–5.

[163] G. Aceto, D. Ciuonzo, A. Montieri, V. Persico, and A. Pescapé, "MIRAGE: Mobile-app traffic capture and ground-truth creation," in *Proc. 4th Int. Conf. Comput., Commun. Secur. (ICCCS)*, Oct. 2019, pp. 1–8.

[164] C. Wang, A. Finamore, L. Yang, K. Fauvel, and D. Rossi, "AppClassNet: A commercial-grade dataset for application identification research," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 52, no. 3, pp. 19–27, Jul. 2022.

[165] M. Ring, S. Wunderlich, D. Scheuring, D. Landes, and A. Hotho, "A survey of network-based intrusion detection data sets," *Comput. Secur.*, vol. 86, pp. 147–167, Sep. 2019.

[166] (2023). *Datasets*. [Online]. Available: https://www.unb.ca/cic/datasets/

[167] A. Dvir, Y. Zion, J. Muehlstein, O. Pele, C. Hajaj, and R. Dubin, "Robust machine learning for encrypted traffic classification," 2016, *arXiv:1603.04865*.

[168] R. Poorzare and O. P. Waldhorst, "Toward the implementation of MPTCP over mmWave 5G and beyond: Analysis, challenges, and solutions," *IEEE Access*, vol. 11, pp. 19534–19566, 2023.

[169] R. Poorzare and A. C. Augé, "Challenges on the way of implementing TCP over 5G networks," *IEEE Access*, vol. 8, pp. 176393–176415, 2020.

[170] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the NS-3 simulator," *SIGCOMM Demonstration*, vol. 14, no. 14, p. 527, 2008.

[171] M. Mezzavilla, M. Zhang, M. Polese, R. Ford, S. Dutta, S. Rangan, and M. Zorzi, "End-to-end simulation of 5G mmWave networks," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 2237–2263, 3rd Quart., 2018.

[172] P. Gawłowicz and A. Zubow, "Ns3-gym: Extending OpenAI gym for networking research," 2018, *arXiv:1810.03943*.

[173] H. Yin, P. Liu, K. Liu, L. Cao, L. Zhang, Y. Gao, and X. Hei, "Ns3-AI: Fostering artificial intelligence algorithms for networking research," in *Proc. Workshop Ns-3*. New York, NY, USA: Association for Computing Machinery, Jun. 2020, pp. 57–64, doi: 10.1145/3389400.3389404.

[174] M. Schettler, D. S. Buse, A. Zubow, and F. Dressler, "How to train your ITS? Integrating machine learning with vehicular network simulation," in *Proc. IEEE Veh. Netw. Conf. (VNC)*, Dec. 2020, pp. 1–4.

[175] D. Stolpmann. (2021). Machine Learning in OMNeT++. GitHub Repository. [Online]. Available: https://github.com/ComNetsHH/omnetpp-ml

[176] "FlowEmu: An open-source flow-based network emulator," *Electron. Commun. EASST*, vol. 80, Sep. 2021.

[177] F. Ruffy, M. Przystupa, and I. Beschastnikh, "Iroko: A framework to prototype reinforcement learning for data center traffic control," 2018, *arXiv:1812.09975*.

[178] J. Charlier, A. Singh, G. Ormazabal, R. State, and H. Schulzrinne, "SynGAN: Towards generating synthetic network attacks using GANs," 2019, *arXiv:1908.09899*.

[179] M. Ring, D. Schlör, D. Landes, and A. Hotho, "Flow-based network traffic generation using generative adversarial networks," *Comput. Secur.*, vol. 82, pp. 156–172, May 2019, doi: 10.1016/j.cose.2018.12.012.

[180] A. Mozo, Á. González-Prieto, A. Pastor, S. Gómez-Canaval, and E. Talavera, "Synthetic flow-based cryptomining attack generation through generative adversarial networks," *Sci. Rep.*, vol. 12, no. 1, p. 2091, Feb. 2022, doi: 10.1038/s41598-022-06057-2.

[181] Y. Guo, G. Xiong, Z. Li, J. Shi, M. Cui, and G. Gou, "Combating imbalance in network traffic classification using GAN based oversampling," in *Proc. IFIP Netw. Conf. (IFIP Networking)*, Jun. 2021, pp. 1–9.

[182] T. J. Anande and M. S. Leeson, "Generative adversarial networks (GANs): A survey on network traffic generation," *Int. J. Mach. Learn. Comput.*, vol. 12, no. 6, pp. 333–343, 2022.

[183] M. Rigaki and S. Garcia, "Bringing a GAN to a knife-fight: Adapting malware communication to avoid detection," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2018, pp. 70–75.

[184] C. Zhang, X. Ouyang, and P. Patras, "ZipNet-GAN: Inferring fine-grained mobile traffic patterns via a generative adversarial neural network," 2017, *arXiv:1711.02413*.

[185] B. Dowoo, Y. Jung, and C. Choi, "PcapGAN: Packet capture file generator by style-based generative adversarial networks," in *Proc. 18th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2019, pp. 1149–1154.

[186] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "Implementation matters in deep RL: A case study on ppo and trpo," in *Proc. Int. Conf. Learn. Represent.*, 2020, pp. 1–14.

[187] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *J. Mach. Learn. Res.*, vol. 22, no. 268, pp. 12348–12355, 2021.

[188] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. Araújo, "CleanRL: High-quality single-file implementations of deep reinforcement learning algorithms," *J. Mach. Learn. Res.*, vol. 23, no. 274, pp. 1–18, 2022. [Online]. Available: http://jmlr.org/papers/v23/21-1342.html

[189] E. Liang, R. Liaw, P. Moritz, R. Nishihara, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica, "RLlib: Abstractions for distributed reinforcement learning," 2017, *arXiv:1712.09381*.

[190] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–14.

[191] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

[192] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," 2018, *arXiv:1812.05905*.

[193] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1587–1596.

[194] H. Mania, A. Guy, and B. Recht, "Simple random search provides a competitive approach to reinforcement learning," 2018, *arXiv:1803.07055*.

[195] W. Dabney, M. Rowland, M. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," in *Proc. AAAI Conf. Artif. Intell.*, 2018, vol. 32, no. 1, pp. 1–10.

[196] S. Huang, R. F. J. Dossa, A. Raffin, A. Kanervisto, and W. Wang, "The 37 implementation details of proximal policy optimization," in *Proc. ICLR Blog Track*, 2023, pp. 1–12.

[197] A. Kuznetsov, P. Shvechikov, A. Grishin, and D. Vetrov, "Controlling overestimation bias with truncated mixture of continuous distributional quantile critics," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 5556–5566.

[198] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proc. 32nd Int. Conf. Mach. Learn.*, vol. 37, Lille, France, 2015, pp. 1889–1897.

[199] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," 2020, *arXiv:2006.14171*.

[200] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 449–458.

[201] K. W. Cobbe, J. Hilton, O. Klimov, and J. Schulman, "Phasic policy gradient," in *Proc. ICML*, 2021, pp. 2020–2027.

[202] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," 2017, *arXiv:1712.01815*.

[203] Q. Wang, J. Xiong, L. Han, H. Liu, and T. Zhang, "Exponentially weighted imitation learning for batched historical data," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 31, 2018, pp. 1–6.

[204] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative Q-learning for offline reinforcement learning," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 1179–1191.

[205] Z. Wang, "Critic regularized regression," in *Proc. Int. Conf. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 7768–7778.

[206] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," 2019, *arXiv:1912.01603*.

[207] L. Espeholt, "IMPALA: Scalable distributed deeP-RL with importance weighted actor-learner architectures," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1407–1416.

[208] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney, "Recurrent experience replay in distributed reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–12.

[209] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Proc. AAAI Conf. Artif. Intell.*, vol. 32, no. 1, 2018, pp. 3215–3222.

[210] E. Ie, V. Jain, J. Wang, S. Narvekar, R. Agarwal, R. Wu, H.-T. Cheng, T. Chandra, and C. Boutilier, "SlateQ: A tractable decomposition for reinforcement learning with recommendation sets," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, pp. 2592–2599.

[211] E. Wijmans, A. Kadian, A. Morcos, S. Lee, I. Essa, D. Parikh, M. Savva, and D. Batra, "DD-PPO: Learning near-perfect PointGoal navigators from 2.5 billion frames," 2019, *arXiv:1911.00357*.

[212] TensorFlow, "Tensorboard: A unified platform for visualizing live, rich data for tensorflow models," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR) Workshops*, Jun. 2016.

[213] L. Biewald. (2020). *Experiment Tracking With Weights and Biases*. [Online]. Available: https://www.wandb.com/

[214] Comet.ml. (2018). *Comet.ML: Machine Learning Operations Platform*. [Online]. Available: https://www.comet.ml/

[215] A. Chen, A. Chow, A. Davidson, A. DCunha, A. Ghodsi, S. A. Hong, A. Konwinski, C. Mewald, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, A. Singh, F. Xie, M. Zaharia, R. Zang, J. Zheng, and C. Zumar, "Developments in MLflow: A system to accelerate the machine learning lifecycle," in *Proc. 4th Int. Workshop Data Manage. End End Mach. Learn.* New York, NY, USA: Association for Computing Machinery, Jun. 2020, pp. 1–4, doi: 10.1145/3399579.3399867.

[216] I. Habibie, M. Kleinsorge, Z. Al-Ars, J. Schneider, W. Kessler, and T. Kuhlen, "Visdom: A tool for visualization and monitoring of machine learning experiments," Tech. Rep., Mar. 2017.

[217] Microsoft. (2021). *Tensorwatch*. GitHub repository. [Online]. Available: https://github.com/microsoft/tensorwatch

[218] M. R. Asia. (2021). *NNI (Neural Network Intelligence): An Open-Source Automl Toolkit for Neural Architecture Search and Hyper-parameter Tuning*. GitHub repository. [Online]. Available: https://github.com/microsoft/nni

[219] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 2623–2631.

[220] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica, "Tune: A research platform for distributed model selection and training," 2018, *arXiv:1807.05118*.

[221] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, and K. Kavukcuoglu, "Population based training of neural networks," 2017, *arXiv:1711.09846*.

[222] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proc. 30th Int. Conf. Mach. Learn.*, vol. 28, S. Dasgupta and D. McAllester, Eds. Atlanta, GA, USA: PMLR, Feb. 2013, pp. 115–123. [Online]. Available: https://proceedings.mlr.press/v28/bergstra13.html

[223] R. Ostrovskiy and A. Gordon. (2020). *Keras Tuner*. GitHub repository. [Online]. Available: https://github.com/keras-team/keras-tuner

[224] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 2546–2554. [Online]. Available: http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization

[225] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2962–2970. [Online]. Available: http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning

[226] M. Merenda, C. Porcaro, and D. Iero, "Edge machine learning for AI-enabled IoT devices: A review," *Sensors*, vol. 20, no. 9, p. 2533, Apr. 2020.

[227] A.-S. Tonneau, N. Mitton, and J. Vandaele, "A survey on (mobile) wireless sensor network experimentation testbeds," in *Proc. IEEE Int. Conf. Distrib. Comput. Sensor Syst.*, May 2014, pp. 263–268.

[228] M. Chernyshev, Z. Baig, O. Bello, and S. Zeadally, "Internet of Things (IoT): Research, simulators, and testbeds," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 1637–1647, Jun. 2018.

[229] S. Zhu, S. Yang, X. Gou, Y. Xu, T. Zhang, and Y. Wan, "Survey of testing methods and testbed development concerning Internet of Things," *Wireless Pers. Commun.*, vol. 123, no. 1, pp. 165–194, Mar. 2022.

[230] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel, "FlockLab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems," in *Proc. ACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, Apr. 2013, pp. 153–165.

[231] R. Trüb, R. Da Forno, L. Daschinger, A. Biri, J. Beutel, and L. Thiele, "Non-intrusive distributed tracing of wireless IoT devices with the FlockLab 2 testbed," *ACM Trans. Internet Things*, vol. 3, no. 1, pp. 1–31, 2021.

[232] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, and T. Watteyne, "FIT IoT-LAB: A large scale open experimental IoT testbed," in *Proc. IEEE 2nd World Forum Internet Things (WF-IoT)*, Dec. 2015, pp. 459–464.

[233] M. Schuß, C. A. Boano, M. Weber, and K. Römer, "A competition to push the dependability of low-power wireless protocols to the edge," in *Proc. 14th EWSN Conf.*, 2017, pp. 54–65.

[234] D. Molteni, G. P. Picco, M. Trobinger, and D. Vecchia, "Cloves: A large-scale ultra-wideband testbed," in *Proc. 20th ACM Conf. Embedded Netw. Sensor Syst.* New York, NY, USA: Association for Computing Machinery, Nov. 2022, pp. 808–809, doi: 10.1145/3560905.3568072.

[235] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "PlanetLab: An overlay testbed for broad-coverage services," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 3, pp. 3–12, Jul. 2003, doi: 10.1145/956993.956995.

[236] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, "An integrated experimental environment for distributed systems and networks," *ACM SIGOPS Operating Syst. Rev.*, vol. 36, pp. 255–270, Dec. 2002.

[237] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "GENI: A federated testbed for innovative network experiments," *Comput. Netw.*, vol. 61, pp. 5–23, Mar. 2014.

[238] L. Yang, F. Wen, J. Cao, and Z. Wang, "EdgeTB: A hybrid testbed for distributed machine learning at the edge with high fidelity," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 10, pp. 2540–2553, Oct. 2022.

[239] F. Hussain, R. Hussain, and E. Hossain, "Explainable artificial intelligence (XAI): An engineering perspective," 2021, *arXiv:2101.03613*.

[240] S. Mukherjee, J. Rupe, and J. Zhu, "XAI for communication networks," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops (ISSREW)*, Oct. 2022, pp. 359–364.

[241] C. Liaskos, S. Nie, A. Tsioliaridou, A. Pitsillides, S. Ioannidis, and I. Akyildiz, "End-to-end wireless path deployment with intelligent surfaces using interpretable neural networks," *IEEE Trans. Commun.*, vol. 68, no. 11, pp. 6792–6806, Nov. 2020.

[242] A.-D. Marcu, S. K. G. Peesapati, J. M. Cortes, S. Imtiaz, and J. Gross, "Explainable artificial intelligence for energy-efficient radio resource management," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Mar. 2023, pp. 1–6.

[243] P. Barnard, I. Macaluso, N. Marchetti, and L. A. DaSilva, "Resource reservation in sliced networks: An explainable artificial intelligence (XAI) approach," in *Proc. IEEE Int. Conf. Commun.*, May 2022, pp. 1530–1535.

[244] A. Palaios, C. L. Vielhaus, D. F. Klzer, C. Watermann, R. Hernangomez, S. Partani, P. Geuer, A. Krause, R. Sattiraju, M. Kasparick, G. Fettweis, F. H. P. Fitzek, H. D. Schotten, and S. Stanczak, "The story of QoS prediction in vehicular communication: From radio environment statistics to network-access throughput prediction," Tech. Rep., 2023. [Online]. Available: https://arxiv.org/abs/2302.11966

[245] S. Hariharan, A. Velicheti, A. S. Anagha, C. Thomas, and N. Balakrishnan, "Explainable artificial intelligence in cybersecurity: A brief review," in *Proc. 4th Int. Conf. Secur. Privacy (ISEA-ISAP)*, Oct. 2021, pp. 1–12.

[246] N. Capuano, G. Fenza, V. Loia, and C. Stanzione, "Explainable artificial intelligence in CyberSecurity: A survey," *IEEE Access*, vol. 10, pp. 93575–93600, 2022.

[247] C. Molnar, *Interpretable Machine Learning*, 2nd ed., 2022. [Online]. Available: https://christophm.github.io/interpretable-ml-book and https://www.amazon.de/Interpretable-Machine-Learning-Making-Explainable/dp/B09TMWHVB4

[248] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI," *Inf. Fusion*, vol. 58, pp. 82–115, Jun. 2020.

[249] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," 2013, *arXiv:1312.6034*.

[250] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Mach. Intell.*, vol. 1, no. 5, pp. 206–215, May 2019.

[251] T. Shapira and Y. Shavitt, "FlowPic: Encrypted Internet traffic classification is as easy as image recognition," in *Proc. IEEE INFOCOM Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2019, pp. 680–687.

[252] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–7.

[253] M. T. Ribeiro, S. Singh, and C. Guestrin, "'Why should I trust you?' explaining the predictions of any classifier," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 1135–1144.

[254] H. Nori, S. Jenkins, P. Koch, and R. Caruana, "InterpretML: A unified framework for machine learning interpretability," 2019, *arXiv:1909.09223*.

[255] R. Agarwal, "Neural additive models: Interpretable machine learning with neural nets," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34. Red Hook, NY, USA: Curran Associates, 2021, pp. 4699–4711.

[256] N. Wehner, A. Seufert, T. Hoßfeld, and M. Seufert, "Explainable data-driven QoE modelling with XAI," in *Proc. 15th Int. Conf. Quality Multimedia Exper. (QoMEX)*, Jun. 2023, pp. 7–12.

[257] K. Brunnström, S. A. Beker, K. De Moor, A. Dooms, S. Egger, M.-N. Garcia, T. Hossfeld, S. Jumisko-Pyykkö, C. Keimel, and M.-C. Larabi, "Qualinet white paper on definitions of quality of experience," Tech. Rep., 2013. [Online]. Available: https://hal.science/hal-00977812

[258] N. Wehner, M. Seufert, J. Schuler, S. Wassermann, P. Casas, and T. Hossfeld, "Improving web QoE monitoring for encrypted network traffic through time series modeling," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 48, no. 4, pp. 37–40, May 2021.

[259] E. Hüllermeier and W. Waegeman, "Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods," *Mach. Learn.*, vol. 110, no. 3, pp. 457–506, Mar. 2021.

[260] A. F. Psaros, X. Meng, Z. Zou, L. Guo, and G. E. Karniadakis, "Uncertainty quantification in scientific machine learning: Methods, metrics, and comparisons," *J. Comput. Phys.*, vol. 477, Mar. 2023, Art. no. 111902.

[261] A. Kendall and Y. Gal, "What uncertainties do we need in Bayesian deep learning for computer vision?" in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–5.

[262] V. Dignum, *Responsible Artificial Intelligence: How To Develop and Use AI in a Responsible Way*, vol. 2156. Cham, Switzerland: Springer, 2019.

[263] Y. Siriwardhana, P. Porambage, M. Liyanage, and M. Ylianttila, "AI and 6G security: Opportunities and challenges," in *Proc. Joint Eur. Conf. Netw. Commun. 6G Summit (EuCNC/6G Summit)*, Jun. 2021, pp. 616–621.

[264] Q. Lu, L. Zhu, X. Xu, J. Whittle, D. Zowghi, and A. Jacquet, "Responsible AI pattern catalogue: A collection of best practices for AI governance and engineering," *ACM Comput. Surv.*, vol. 56, no. 7, pp. 1–35, Oct. 2023, doi: 10.1145/3626234.

[265] W. Yang, H. Le, T. Laud, S. Savarese, and S. C. H. Hoi, "OmniXAI: A library for explainable AI," 2022, *arXiv:2206.01612*.

[266] V. Arya, R. K. E. Bellamy, P.-Y. Chen, A. Dhurandhar, M. Hind, S. C. Hoffman, S. Houde, Q. V. Liao, R. Luss, A. Mojsilović, S. Mourad, P. Pedemonte, R. Raghavendra, J. Richards, P. Sattigeri, K. Shanmugam, M. Singh, K. R. Varshney, D. Wei, and Y. Zhang, "AI explainability 360 toolkit," in *Proc. 3rd ACM India Joint Int. Conf. Data Sci. Manage. Data (8th ACM IKDD CODS 26th COMAD)*, Jan. 2021, pp. 376–379.

[267] J. Klaise, A. Van Looveren, G. Vacanti, and A. Coca, "Alibi explain: Algorithms for explaining machine learning models," *J. Mach. Learn. Res.*, vol. 22, no. 1, pp. 8194–8200, 2021.

[268] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *Proc. Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 3319–3328.

[269] N. Kokhlikyan, V. Miglani, M. Martin, E. Wang, B. Alsallakh, J. Reynolds, A. Melnikov, N. Kliushkina, C. Araya, S. Yan, and O. Reblitz-Richardson, "Captum: A unified and generic model interpretability library for PyTorch," 2020, *arXiv:2009.07896*.

[270] (2019). *TorchRay*. [Online]. Available: https://github.com/facebookresearch/TorchRay

[271] (2019). *TF-Explain*. [Online]. Available: https://tf-explain.readthedocs.io/en/latest/index.html

[272] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2224–2287, 3rd Quart., 2019.

[273] H. Hellström, J. Mairton B. da Silva Jr., M. M. Amiri, M. Chen, V. Fodor, H. V. Poor, and C. Fischione, "Wireless for machine learning," 2020, *arXiv:2008.13492*.

[274] D. Jin, Z. Yu, P. Jiao, S. Pan, D. He, J. Wu, P. S. Yu, and W. Zhang, "A survey of community detection approaches: From statistical modeling to deep learning," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 2, pp. 1149–1170, Feb. 2023.

[275] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," 2018, *arXiv:1812.08434*.

[276] M. A. Ridwan, N. A. M. Radzi, F. Abdullah, and Y. E. Jalil, "Applications of machine learning in networking: A survey of current issues and future challenges," *IEEE Access*, vol. 9, pp. 52523–52556, 2021.

[277] F. Tang, B. Mao, N. Kato, and G. Gui, "Comprehensive survey on machine learning in vehicular network: Technology, applications and challenges," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 3, pp. 2027–2057, 3rd Quart., 2021.

[278] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn, "Estimation of energy consumption in machine learning," *J. Parallel Distrib. Comput.*, vol. 134, pp. 75–88, Dec. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731518308773

[279] L. Song, X. Hu, G. Zhang, P. Spachos, K. N. Plataniotis, and H. Wu, "Networking systems of AI: On the convergence of computing and communications," *IEEE Internet Things J.*, vol. 9, no. 20, pp. 20352–20381, Oct. 2022.

[280] G. Drainakis, K. V. Katsaros, P. Pantazopoulos, V. Sourlas, and A. Amditis, "Federated vs. centralized machine learning under privacy-elastic users: A comparative analysis," in *Proc. IEEE 19th Int. Symp. Netw. Comput. Appl. (NCA)*, Nov. 2020, pp. 1–8.

[281] I. A. Majeed, S. Kaushik, A. Bardhan, V. S. K. Tadi, H.-K. Min, K. Kumaraguru, and R. D. Muni, "Comparative assessment of federated and centralized machine learning," 2022, *arXiv:2202.01529*.

[282] W. Hassan, T.-S. Chou, O. Tamer, J. Pickard, P. Appiah-Kubi, and L. Pagliari, "Cloud computing survey on services, enhancements and challenges in the era of machine learning and data science," *Int. J. Informat. Commun. Technol. (IJ-ICT)*, vol. 9, no. 2, p. 117, Aug. 2020.

[283] Y. Ko, K. Choi, H. Jei, D. Lee, and S.-W. Kim, "ALADDIN: Asymmetric centralized training for distributed deep learning," in *Proc. 30th ACM Int. Conf. Inf. Knowl. Manage.*, Oct. 2021, pp. 863–872.
[284] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 869–904, 2nd Quart., 2020.
[285] F. Samie, L. Bauer, and J. Henkel, "From cloud down to things: An overview of machine learning in Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4921–4934, Jun. 2019.
[286] W. Toussaint and A. Y. Ding, "Machine learning systems in the IoT: Trustworthiness trade-offs for edge intelligence," in *Proc. IEEE 2nd Int. Conf. Cognit. Mach. Intell. (CogMI)*, Oct. 2020, pp. 177–184.
[287] A. Smola and S. Narayanamurthy, "An architecture for parallel topic models," *Proc. VLDB Endowment*, vol. 3, nos. 1–2, pp. 703–710, Sep. 2010.
[288] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *Proc. 11th USENIX Symp. Operating Syst. Des. Implement. (OSDI)*, 2014, pp. 583–598.
[289] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012, pp. 1–8.
[290] J. Jiang, B. Cui, C. Zhang, and L. Yu, "Heterogeneity-aware distributed parameter servers," in *Proc. ACM Int. Conf. Manage. Data*. New York, NY, USA: Association for Computing Machinery, May 2017, pp. 463–478.
[291] B. McMahan, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1273–1282.
[292] Y. Liu, Y. Kang, C. Xing, T. Chen, and Q. Yang, "A secure federated transfer learning framework," *IEEE Intell. Syst.*, vol. 35, no. 4, pp. 70–82, Jul. 2020.
[293] F. Chen, M. Luo, Z. Dong, Z. Li, and X. He, "Federated meta-learning with fast convergence and efficient communication," 2018, *arXiv:1802.07876*.
[294] A. Gibiansky, "Bringing HPC techniques to deep learning," Baidu Res., Beijing, China, Tech. Rep., 2017.
[295] P. Patarasuk and X. Yuan, "Bandwidth optimal all-reduce algorithms for clusters of workstations," in *Proc. J. Parallel Distrib. Comput.*, Feb. 2009, vol. 69, no. 2, pp. 117–124.
[296] H. Zhao and J. Canny, "Butterfly mixing: Accelerating incremental-update algorithms on clusters," in *Proc. SIAM Int. Conf. Data Mining*, May 2013, pp. 785–793.
[297] X. Wan, H. Zhang, H. Wang, S. Hu, J. Zhang, and K. Chen, "RAT–Resilient allreduce tree for distributed machine learning," in *Proc. 4th Asia–Pacific Workshop Netw.*, Aug. 2020, pp. 52–57.
[298] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *J. Netw. Comput. Appl.*, vol. 116, pp. 1–8, Aug. 2018.
[299] E. Samikwa, A. D. Maio, and T. Braun, "ARES: Adaptive resource-aware split learning for Internet of Things," *Comput. Netw.*, vol. 218, Dec. 2022, Art. no. 109380.
[300] V. Turina, Z. Zhang, F. Esposito, and I. Matta, "Federated or split? A performance and privacy analysis of hybrid split and federated learning architectures," in *Proc. IEEE 14th Int. Conf. Cloud Comput. (CLOUD)*, Sep. 2021, pp. 250–260.
[301] Y. Gao, M. Kim, C. Thapa, A. Abuadbba, Z. Zhang, S. Camtepe, H. Kim, and S. Nepal, "Evaluation and optimization of distributed machine learning techniques for Internet of Things," *IEEE Trans. Comput.*, vol. 71, no. 10, pp. 2538–2552, Oct. 2022.
[302] E. Samikwa, A. Di Maio, and T. Braun, "Adaptive early exit of computation for energy-efficient and low-latency machine learning over IoT networks," in *Proc. IEEE 19th Annu. Consum. Commun. Netw. Conf. (CCNC)*, Jan. 2022, pp. 200–206.
[303] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Comput. Surv.*, vol. 55, no. 5, pp. 1–30, May 2023.
[304] Y. Matsubara, R. Yang, M. Levorato, and S. Mandt, "SC2 benchmark: Supervised compression for split computing," 2022, *arXiv:2203.08875*.
[305] M. G. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," *ACM Comput. Surveys*, vol. 54, no. 8, pp. 1–37, Oct. 2021, doi: 10.1145/3469029.
[306] J. Shuja, K. Bilal, W. Alasmary, H. Sinky, and E. Alanazi, "Applying machine learning techniques for caching in next-generation edge networks: A comprehensive survey," *J. Netw. Comput. Appl.*, vol. 181, May 2021, Art. no. 103005.
[307] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu, "A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 1, pp. 393–430, 1st Quart., 2019.
[308] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo, "A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities," *J. Internet Services Appl.*, vol. 9, no. 1, pp. 1–99, Dec. 2018.
[309] Y. Liu, F. R. Yu, X. Li, H. Ji, and V. C. M. Leung, "Blockchain and machine learning for communications and networking systems," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 1392–1431, 2nd Quart., 2020.
[310] O. Nassef, W. Sun, H. Purmehdi, M. Tatipamula, and T. Mahmoodi, "A survey: Distributed machine learning for 5G and beyond," *Comput. Netw.*, vol. 207, Apr. 2022, Art. no. 108820.
[311] O. A. Wahab, A. Mourad, H. Otrok, and T. Taleb, "Federated machine learning: Survey, multi-level classification, desirable criteria and future directions in communication and networking systems," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 2, pp. 1342–1397, 2nd Quart., 2021.
[312] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 2031–2063, 3rd Quart., 2020.
[313] A. Imteaj, K. Mamun Ahmed, U. Thakker, S. Wang, J. Li, and M. H. Amini, "Federated learning for resource-constrained IoT devices: Panoramas and state of the art," in *Federated and Transfer Learning*. Cham, Switzerland: Springer, 2022, pp. 7–27.
[314] M. Abbasi, A. Shahraki, and A. Taherkordi, "Deep learning for network traffic monitoring and analysis (NTMA): A survey," *Comput. Commun.*, vol. 170, pp. 19–41, Mar. 2021.
[315] F. Hussain, S. A. Hassan, R. Hussain, and E. Hossain, "Machine learning for resource management in cellular and IoT networks: Potentials, current solutions, and open challenges," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 1251–1275, 2nd Quart., 2020.
[316] A. Talpur and M. Gurusamy, "Machine learning for security in vehicular networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 1, pp. 346–379, 1st Quart., 2022.
[317] M. A. Hossain, R. M. Noor, K. A. Yau, S. R. Azzuhri, M. R. Z'aba, and I. Ahmedy, "Comprehensive survey of machine learning approaches in cognitive radio-based vehicular ad hoc networks," *IEEE Access*, vol. 8, pp. 78054–78108, 2020.
[318] W. Guo, "Explainable artificial intelligence for 6G: Improving trust between human and machine," *IEEE Commun. Mag.*, vol. 58, no. 6, pp. 39–45, Jun. 2020.
[319] Y. Zheng, Z. Liu, X. You, Y. Xu, and J. Jiang, "Demystifying deep learning in networking," in *Proc. 2nd Asia–Pacific Workshop Netw.*, Aug. 2018, pp. 1–7.
[320] A. Adadi and M. Berrada, "Peeking inside the black-box: A survey on explainable artificial intelligence (XAI)," *IEEE Access*, vol. 6, pp. 52138–52160, 2018.
[321] A. Heuillet, F. Couthouis, and N. Díaz-Rodríguez, "Explainability in deep reinforcement learning," *Knowl.-Based Syst.*, vol. 214, Feb. 2021, Art. no. 106685.
[322] A. Paleyes, R.-G. Urma, and N. D. Lawrence, "Challenges in deploying machine learning: A survey of case studies," *ACM Comput. Surv.*, vol. 55, no. 6, pp. 1–29, Jul. 2023.
[323] S. Faezi and A. Shirmarz, "A comprehensive survey on machine learning using in software defined networks (SDN)," *Hum.-Centric Intell. Syst.*, vol. 3, no. 3, pp. 312–343, Jun. 2023.
[324] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for SDN? Implementation challenges for software-defined networks," *IEEE Commun. Mag.*, vol. 51, no. 7, pp. 36–43, Jul. 2013.
[325] C. Lu, A. Saifullah, B. Li, M. Sha, H. Gonzalez, D. Gunatilaka, C. Wu, L. Nie, and Y. Chen, "Real-time wireless sensor-actuator networks for industrial cyber-physical systems," *Proc. IEEE*, vol. 104, no. 5, pp. 1013–1024, May 2016.

[326] H. Kopetz and W. Steiner, "Real-time communication," in *Real-time Systems: Design Principles for Distributed Embedded Applications*. Cham, Switzerland: Springer, 2022, pp. 177–200.

[327] D. Zhang, P. Shi, Q.-G. Wang, and L. Yu, "Analysis and synthesis of networked control systems: A survey of recent advances and challenges," *ISA Trans.*, vol. 66, pp. 376–392, Jan. 2017.

[328] S. Ramstedt and C. Pal, "Real-time reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 32, 2019, pp. 1–9.

[329] J. Mendez, K. Bierzynski, M. P. Cuéllar, and D. P. Morales, "Edge intelligence: Concepts, architectures, applications, and future directions," *ACM Trans. Embedded Comput. Syst.*, vol. 21, no. 5, pp. 1–41, Sep. 2022.

[330] L. E. Lwakatare, A. Raj, I. Crnkovic, J. Bosch, and H. H. Olsson, "Large-scale machine learning systems in real-world industrial settings: A review of challenges and solutions," *Inf. Softw. Technol.*, vol. 127, Nov. 2020, Art. no. 106368.

[331] A. Redder, A. Ramaswamy, and H. Karl, "Stability and convergence of distributed stochastic approximations with large unbounded stochastic information delays," 2023, *arXiv:2305.07091*.

[332] R. Bless, B. Bloessl, M. Hollick, M. Corici, H. Karl, D. Krummacker, D. Lindenschmitt, H. D. Schotten, and L. Wimmer, "Dynamic network (re-)configuration across time, scope, and structure," in *Proc. Joint Eur. Conf. Netw. Commun. 6G Summit (EuCNC/6G Summit)*, Jun. 2022, pp. 547–552.

[333] J. Hoydis, F. A. Aoudia, A. Valcarce, and H. Viswanathan, "Toward a 6G AI-native air interface," *IEEE Commun. Mag.*, vol. 59, no. 5, pp. 76–81, May 2021.

[334] F. Ait Aoudia, J. Hoydis, A. Valcarce, and H. Viswanathan. (2021). *Toward a 6G AI-Native Air Interface*. [Online]. Available: https://www.bell-labs.com/institute/white-papers/toward-6g-ai-native-air-interface/

[335] R. Schwarz. (2023). *Enabling an AI-Native Air Interface for 6G: Rohde & Schwarz Showcases AI/ML-Based Neural Receiver With Optimized Modulation at Brooklyn 6G Summit, in Collaboration With Nvidia*. [Online]. Available: https://www.rohde-schwarz.com/se/about/news-press/all-news/enabling-an-ai-native-air-interface-for-6g-rohde-schwarz-showcases-ai-ml-based-neural-receiver-with-optimized-modulation-at-brooklyn-6g-summit-in-collaboration-with-nvidia-press-release-detailpage229356-1425541.html

[336] Ericsson. (2021). *Defining AI Native: A Key Enabler for Advanced Intelligent Telecom Networks*. [Online]. Available: https://www.ericsson.com/en/reports-and-papers/white-papers/ai-native

[337] C. Chaccour, W. Saad, M. Debbah, Z. Han, and H. V. Poor, "Less data, more knowledge: Building next generation semantic communication networks," Tech. Rep., 2022. [Online]. Available: https://arxiv.org/abs/2211.14343

[338] C. K. Thomas, C. Chaccour, W. Saad, M. Debbah, and C. S. Hong, "Causal reasoning: Charting a revolutionary course for next-generation AI-native wireless networks," *IEEE Veh. Technol. Mag.*, vol. 19, no. 1, pp. 16–31, Mar. 2024.

[339] A. Mughees, M. Tahir, M. A. Sheikh, and A. Ahad, "Towards energy efficient 5G networks using machine learning: Taxonomy, research challenges, and future research directions," *IEEE Access*, vol. 8, pp. 187498–187522, 2020.

[340] D. Li, X. Chen, M. Becchi, and Z. Zong, "Evaluating the energy efficiency of deep convolutional neural networks on CPUs and GPUs," in *Proc. IEEE Int. Conf. Big Data Cloud Comput. (BDCloud), Social Comput. Netw. (SocialCom), Sustain. Comput. Commun. (SustainCom) (BDCloud-SocialCom-SustainCom)*, Oct. 2016, pp. 477–484.

[341] M. Svedin, S. W. D. Chien, G. Chikafa, N. Jansson, and A. Podobas, "Benchmarking the Nvidia GPU lineage: From early K80 to modern A100 with asynchronous memory transfers," in *Proc. 11th Int. Symp. Highly Efficient Accel. Reconfigurable Technol.*, Jun. 2021, pp. 1–6.

[342] E. Samikwa, A. D. Maio, and T. Braun, "DISNET: Distributed micro-split deep learning in heterogeneous dynamic IoT," *IEEE Internet Things J.*, vol. 11, no. 4, pp. 6199–6216, Feb. 2024.

[343] Y. Chen, T.-J. Yang, J. Emer, and V. Sze, "Understanding the limitations of existing energy-efficient design approaches for deep neural networks," *Energy*, vol. 2, no. L1, p. L3, 2018.

[344] E. Hossain and F. Fredj, "Editorial energy efficiency of machine-learning-based designs for future wireless systems and networks," *IEEE Trans. Green Commun. Netw.*, vol. 5, no. 3, pp. 1005–1010, Sep. 2021.

[345] R. Desislavov, F. Martínez-Plumed, and J. Hernández-Orallo, "Trends in AI inference energy consumption: Beyond the performance-vs-parameter laws of deep learning," *Sustain. Comput., Informat. Syst.*, vol. 38, Apr. 2023, Art. no. 100857. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210537923000124

[346] T.-J. Yang, Y.-H. Chen, J. Emer, and V. Sze, "A method to estimate the energy consumption of deep neural networks," in *Proc. 51st Asilomar Conf. Signals, Syst., Comput.*, Oct. 2017, pp. 1916–1920.

[347] T.-J. Yang, Y.-H. Chen, and V. Sze, "Deep neural network energy estimation tool," MIT, Tech. Rep., Jan. 2017. Accessed: Jan. 25, 2024. [Online]. Available: https://energyestimation.mit.edu/

[348] J. Lin, W.-M. Chen, Y. Lin, C. Gan, S. Han, and J. Cohn, "MCUNet: Tiny deep learning on IoT devices," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 33, 2020, pp. 11711–11722.

[349] H. Cai, C. Gan, L. Zhu, and S. Han, "TinyTL: Reduce activations, not trainable parameters for efficient on-device learning," 2020, *arXiv:2007.11622*.

[350] L. Heim, A. Biri, Z. Qu, and L. Thiele, "Measuring what really matters: Optimizing neural networks for TinyML," 2021, *arXiv:2104.10645*.

[351] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," 2017, *arXiv:1712.01887*.

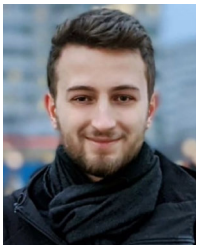
[352] Y. Abadade, A. Temouden, H. Bamoumen, N. Benamar, Y. Chtouki, and A. S. Hafid, "A comprehensive survey on TinyML," *IEEE Access*, vol. 11, pp. 96892–96922, 2023.

HAITHAM AFIFI (Member, IEEE) received the B.Sc. degree in information engineering and technology and the M.Sc. degree in communication engineering from German University in Cairo, in 2014 and 2015, respectively, and the Ph.D. degree from the Hasso Plattner Institute, in 2023. He has industry experience as a Network Engineer with Orange Business Services and an IT Consultant for integrating generative AI in network operations. His research interests include wireless network virtualization and reinforcement learning and network optimization.

SABRINA POCHABA received the master's degree in mathematics from Ruprecht Karls University, Heidelberg, Germany. She is currently pursuing the Ph.D. degree with Paris Lodron University of Salzburg. Since 2021, she has a Data Scientist with Salzburg Research Forschungsgesellschaft. There she is engaged in different machine learning methods, focusing on networks and communication.

ANDREAS BOLTRES received the B.S. and M.S. degrees in informatics from Karlsruhe Institute of Technology (KIT), Germany, in 2017 and 2021, respectively, where he is currently pursuing the Ph.D. degree with the Autonomous Learning Robots Laboratory. His research interests include multi-agent and swarm reinforcement learning and their applications to robotics and computer networking, in particular routing optimization and traffic engineering.

DOMINIC LANIEWSKI (Graduate Student Member, IEEE) received the B.S. degree in information systems from the University of Münster, in 2017, and the M.S. degree in computer science from Osnabrück University, in 2019, where he is currently pursuing the Ph.D. degree with the Distributed Systems Group. His research interests include machine learning for networks, QoE of streaming applications, video and point cloud streaming, and robot communications.



JANEK HABERER received the B.Sc. and M.Sc. degrees in computer science from Kiel University, Germany, in 2019 and 2021, respectively, where he is currently pursuing the Ph.D. degree with the Distributed Systems Group. His research interests include distributed machine learning and its applications in the Internet of Things, particularly edge computing and split learning.

LEONARD PAELEKE received the B.Eng. degree in mechanical engineering from Berliner Hochschule für Technik (BHT), in 2018, and the M.S. degree in computational engineering from TU Berlin, in 2022. Since April 2022, he has been a Ph.D. Researcher with the Internet-Technologies and Softwarization Group, Hasso Plattner Institute (HPI). His research interests include machine learning for networks, networks for machine learning, distributed machine learning, and their application in mobile telecommunication networks, such as 6G.

REZA POORZARE (Member, IEEE) received the B.S. and M.S. degrees in computer engineering from Azad University, Iran, in 2010 and 2014, respectively, and the Ph.D. degree in network engineering from Universitat Politcnica de Catalunya, Barcelona, Spain, in 2022. Currently, he is a Postdoctoral Researcher with the Data-Centric Software Systems (DSS) Research Group, Institute of Applied Research, Karlsruhe University of Applied Sciences, Karlsruhe, Germany. His research interests include 5G, mmWave, wireless mobile networks, TCP, MPTCP, congestion control, and artificial intelligence.

DANIEL STOLPMANN received the B.Sc. and M.Sc. degrees in computer science and engineering from Hamburg University of Technology (TUHH), Germany, in 2017 and 2019, respectively. During the master's studies, he was with the Institute of Communication Networks (ComNets) as a Student Assistant and became a Research Fellow after his graduation. His research interests include machine learning for communication networks, congestion control, active queue management, network coding, and network emulation.

NIKOLAS WEHNER received the master's degree in computer science from the University of Würzburg, Germany. In 2018, he started to work as a Research Engineer with the Center for Technology Experience, AIT—Austrian Institute of Technology, Vienna, Austria. Since October 2019, he has been a Ph.D. Researcher with the Chair of Communication Networks, University of Würzburg. His research interests include QoE of internet applications, machine learning for networks, and user-centric communication networks.

ADRIAN REDDER received the Master of Science degree in electrical engineering from Paderborn University (UPB), Germany, in 2019, with a major in control and information theory, and the Ph.D. degree in computer science with a thesis on distributed stochastic approximation algorithms in April 2024. After Graduate studies, he was a member of the Computer Networks Group at UPB, and since October 2023, he has been a member of the Automatic Control Group at UPB. His research interests include control theory, reinforcement learning, distributed systems, and stochastic approximation algorithms.

ERIC SAMIKWA received the M.Sc. degree in computer science and engineering from the Royal Institute of Technology (KTH), Stockholm, Sweden, in 2020. He is currently pursuing the Ph.D. degree with the Communication and Distributed Systems Group, Institute of Computer Science, University of Bern, Bern, Switzerland. His research interests include distributed machine learning, federated learning, split learning, edge computing, and the Internet of Things.

MICHAEL SEUFERT (Senior Member, IEEE) received the Diploma and Ph.D. degrees in computer science, the bachelor's degree in economathematics, and the Habilitation degree in computer science from the University of Würzburg, Germany, in 2011, 2017, 2018, and 2023, respectively. He also received the First State Examination degree in mathematics, computer science, and education for teaching in secondary schools in 2011. He is a Full Professor with the University of Augsburg, Germany, heading the Chair of Networked Embedded Systems and Communication Systems. His research focuses on user-centric communication networks, including QoE of internet applications, AI/ML for QoE-aware network management, as well as group-based communications.