# Orchestrating Development Lifecycle of Machine Learning Based IoT Applications: A Survey

BIN QIAN, JIE SU, ZHENYU WEN*, DEVKI NANDAN JHA, YINHAO LI, YU GUAN, DEEPAK PUTHAL, and PHILIP JAMES, Newcastle University, UK
RENYU YANG, University of Leeds, UK
ALBERT Y. ZOMAYA, The University of Sydney, Australia
OMER RANA, Cardiff University, UK
LIZHE WANG, China University of Geoscience (Wuhan), China
RAJIV RANJAN, Newcastle University, UK

Machine Learning (ML) and Internet of Things (IoT) are complementary advances: ML techniques unlock complete potentials of IoT with intelligence, and IoT applications increasingly feed data collected by sensors into ML models, thereby employing results to improve their business processes and services. Hence, orchestrating ML pipelines that encompasses model training and implication involved in holistic development lifecycle of an IoT application often leads to complex system integration. This paper provides a comprehensive and systematic survey on the development lifecycle of ML-based IoT application. We outline core roadmap and taxonomy, and subsequently assess and compare existing standard techniques used in individual stage.

Additional Key Words and Phrases: IoT, Machine learning, Deep learning, Orchestration

## 1 INTRODUCTION

Rapid development of hardware, software and communication technologies boosts the speed of connecting the physical world to the Internet via Internet of Things (IoT). A report [1] shows that about 75.44 billion IoT devices will be connected to the Internet by 2025. These massive devices generate a large amount of data with various modalities. Processing and analyzing such big data is essential for developing smart IoT applications. Machine Learning (ML) plays a vital role in data intelligence which aims to understand and explore the real world. ML + IoT type applications thus are experiencing explosive growth. However, there are unfilled gaps between current solutions and demands of orchestrating development lifecycle of ML-based IoT applications. Exiting orchestration frameworks for example *Ubuntu Juju*, *Puppet* and *Chef* are flexible in providing solutions for deploying and running applications over public or private clouds. These frameworks, however, neglect heterogeneity of IoT environment that encompasses various hardwares, communication protocols and operation systems. More importantly, none of them are able to completely orchestrate holistic development lifecycle of ML-based IoT applications. The development lifecycle must cover the following factors: 1) *how* the target application is specified and developed? 2) *where* the target application is deployed? (3) *what* kind of information of the target application needs to be audited. Application specification defines the requirements including the ML tasks, performance, accuracy and execution workflow. Based on the specification and the available computing resources, the ML models are developed to meet the specified requirements while optimizing the training processes in terms of the cost of time and computing resources. Next, the model deployment considers the difficulty of the heterogeneity of the IoT environment

*Zhenyu is the corresponding author
[1] https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

for running a set of composed ML models. Finally, ML-based IoT applications closely connect with people's lives and some applications such as autopilot require high reliability. Therefore, the accountability aims to collect the essential monitoring information to improve the performance of the application in next iteration of the lifecycle.

In this survey, we present a comprehensive research on orchestrating the development lifecycle of ML-based IoT applications. We first present the core roadmap and taxonomy, and subsequently summarize, assess, compare, and assess the variety of techniques used in each step of the lifecycle. Previous efforts provided broad knowledge that can drive us to build the taxonomy. For instance, [251] discussed encountered challenges of developing next generation of AI systems. [179, 302] gave comprehensive reviews of available deep learning architectures and algorithms in IoT domain. To the best of our knowledge, it is the first work that presents a comprehensive survey to illustrate the whole development lifecycle of ML-based IoT application, which paves the way for developing an agile, robust and reliable smart IoT application. Before introducing the roadmap and the taxonomy, we provide a smart city example in next subsection that illustrates the ML-based IoT applications in real-world.

### 1.1 Smart City Applications

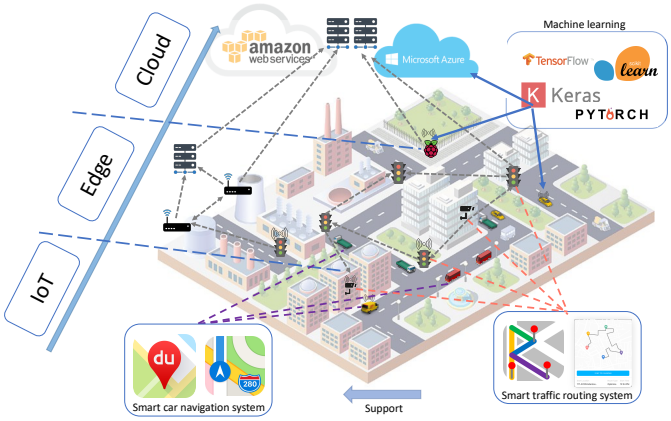


Fig. 1. Smart City

Smart city uses modern communication and information techniques to monitor, integrate and analyze the data collected from core systems running across cities. Meanwhile, smart city makes intelligent responses to various use cases, such as traffic control, weather forecasting, industrial and commercial activities. Fig 1 demonstrates a miniature of the smart city which consists of various IoT applications with many of them using Machine Learning (ML) techniques. For example, a *smart traffic routing* system consists of a large number of cameras monitoring the road traffic and a smart algorithm running on the cloud recommending the optimal routes for users [314]. On the other hand, a *smart car navigation system* [119] allows the passengers to set and change destinations via built-in car audio devices. The two systems work in pair to provide real-time interactive routing services. More specifically, the user's voice commands are translated in the car edge side and sent to the cloud where the *smart traffic routing* system works. The best route is translated back to voice guiding the passengers to their destinations. The above-mentioned applications involve various computing resources(e.g., cloud, edge, and IoT devices) and ML techniques, making the development of these ML-based IoT applications very challenging both for the ML models and the IoT system. To fill this gap, we orchestrate the development lifecycle of an ML-based IoT application. In the next subsection, we present a roadmap for the development lifecycle along with a comprehensive taxonomy that surveys the techniques relevant for developing the application.

### 1.2 Roadmap and Taxonomy

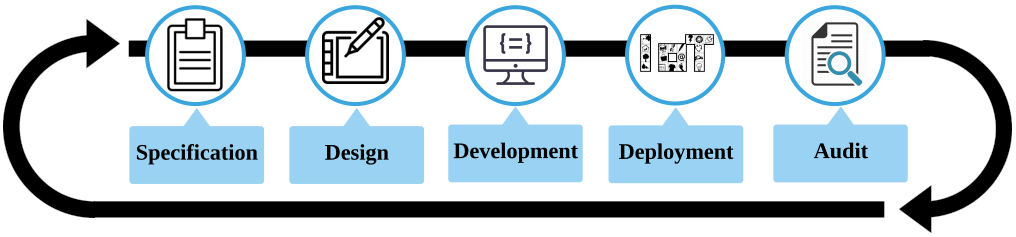


Fig. 2. The development lifecycle of ML-based IoT application

**Roadmap.** Fig 2 shows the roadmap of developing an ML-based IoT application. The roadmap starts with the requirements specification where the required computing resources (hardware and software) and ML models are specified. Based on the specification, we carefully design the infrastructure protocol, data acquisition approach and machine learning model development pipeline. Next, we implement and train the model with various ML algorithms. We also evaluate and optimize the models to achieve high efficiency, without sacrificing too much accuracy. After the model development, an optimized deployment plan is generated based on the specified ML models and infrastructures. The deployed application must be audited while it is running on real IoT environments; the audit aims to explore the performance issues in terms of security, reliability and other QoS metrics. Finally, the audited issues will guide the corrections of orchestration details in the next iteration of the application development.

**Taxonomy.** Fig 3 depicts our taxonomy that systematically analyzes the core components in the orchestration of the development lifecycle of a ML-based IoT application. Note that the survey in [280] has reviewed cloud resource orchestration techniques. It outlines the key infrastructure orchestration challenges for cloud based application as well as extendable for IoT applications. Thus, in this survey, we focus more on the challenges of implementing ML models and orchestrating their IoT application development lifecycle. To this end, we extract the core building blocks of the development lifecycle relevant to ML and identify *four* main categories based on their specific functionality during the development process. The outline of the paper follows the structure of the taxonomy as well.

(1) **Model Development.** We propose a general pipeline for developing a ready-to-deploy ML model. We investigate the ML techniques to build each block of the pipeline (refer to §2).
(2) **Model Deployment.** In our work, we review the software deployment techniques and analyze the challenges of applying such techniques to deploy the ML models in IoT environments (refer to §3).
(3) **Model Audit.** Audit is one of the important dimensions in building a robust application. We survey the main security, reliability and performance issues in the ML-based IoT applications (refer to §4).
(4) **Data Acquisition.** Data quality is important in building the ML models. We identify three dimensions that are important throughout the data acquisition pipeline: data collection, data fusion and data preprocessing (refer to §5).

## 2 MODEL DEVELOPMENT

One of the core components in this paper is machine learning (ML) models, which may be roughly divided into 3 categories: Traditional Machine Learning (TML), Deep Learning (DL) and Reinforcement Learning (RL). To develop

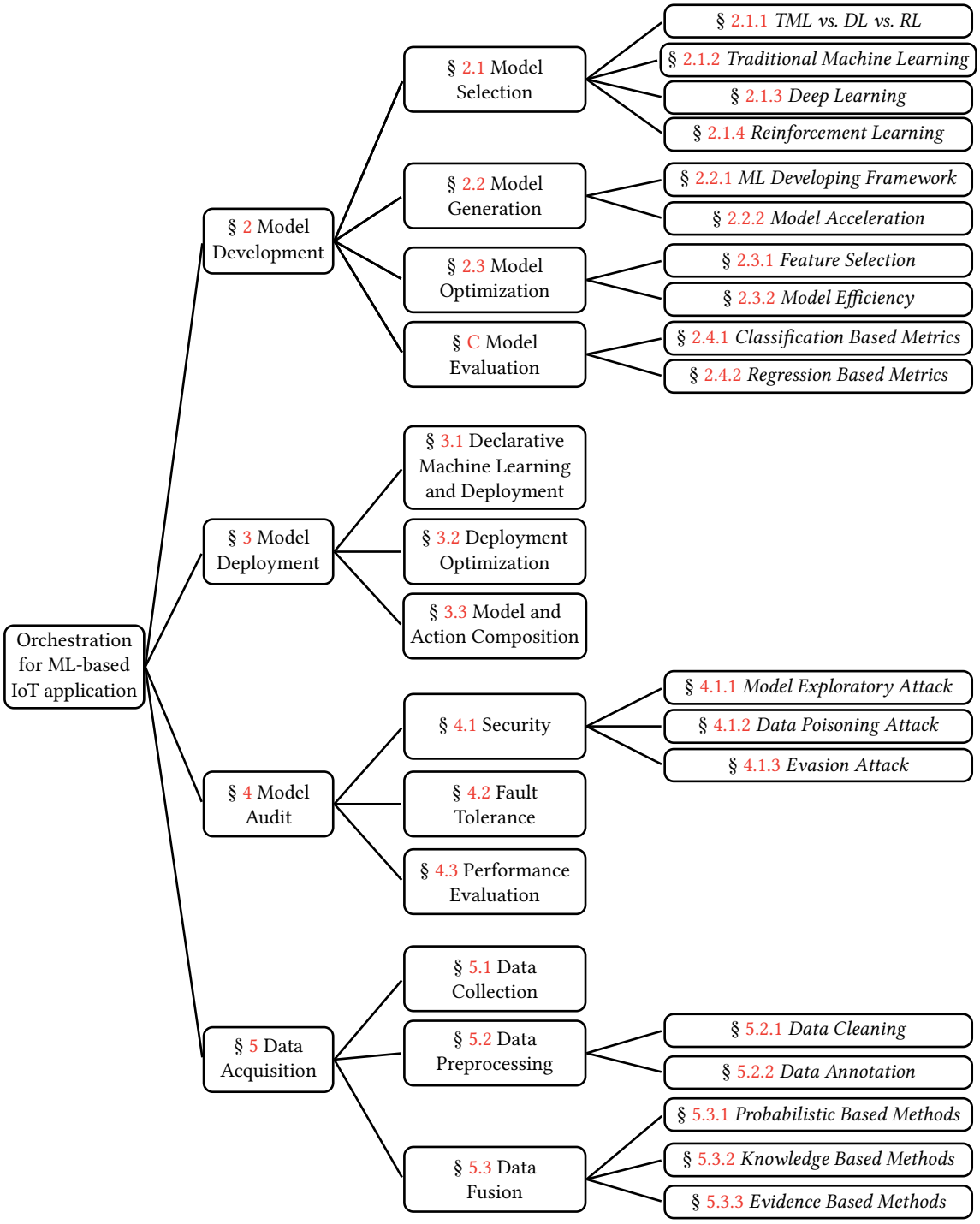


Fig. 3. A taxonomy for orchestrating ML-based IoT application development lifecycle

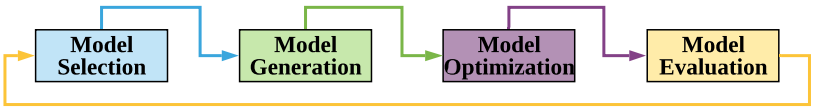


Fig. 4. A general pipeline of model development

ML models in the IoT environment, we propose a generic pipeline (see Fig. 4) including *model selection*, *model generation*, *model optimization* and *model evaluation*. We initially introduce generic pipeline by presenting adaptive video streaming [168] as an example.

**Adaptive video streaming.** Video transmission between server and mobile devices employ http-based adaptive streaming techniques. In a typical video server (e.g., DASH [2]), videos are encoded and stored as multiple chunks at different bitrates. One video usually consists of several chunks with each containing seconds of content. To maximize video quality, video player in client (e.g., mobile device) usually employs adaptive bitrate (ABR) algorithms aiming at pulling high-bitrate chunks from the server without compromising the latency. As shown in Fig. 5, ABR algorithms use simple heuristics to make bitrate decisions based on various observations such as the estimated network throughput and playback buffer occupancy. ABR algorithms require fine-grained tuning and can be hardly generalized to handle various network conditions that fluctuate across time and different environments. Thus we are seeking to solve the problem using modern ML technologies.

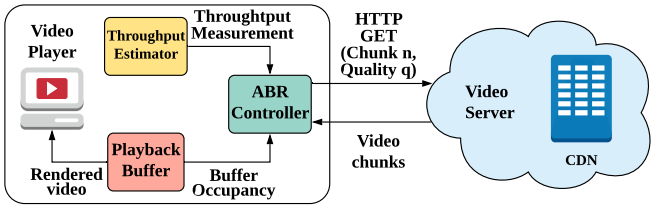


Fig. 5. Adaptive video streaming

To this end, we first need to perform **model selection** (§2.1) to find a subset of suitable models. In the scenario, the server must give a bitrate decision so that the client can return a feedback that conveys whether the decision is satisfactory. Such kind of interaction problems necessitate further use of RL and we will present proper choice of RL algorithms based on different selecting criteria (§2.1.4). Next, we will choose a suitable development framework to implement the model and utilize some acceleration techniques (§2.2.2) to reduce the latency of **model generation** (§2.2). In this example, Tensorflow and A3C algorithm [177] are used as the development framework and distributed training protocol respectively for faster convergence. Once generated, the model has to be adapted into the real environment. Considering heterogeneity of IoT infrastructure, models need to be optimized according to the computing resources. This procedure is called **model optimization** (§2.3). In **model evaluation** (§C), model performance is measured to validate if the model meets expected results. Particularly in this case, performance is evaluated by the total reward obtained from the simulated environment. Following subsections will discuss the the pipeline in details.

### 2.1 Model Selection

Model selection aims to find the *optimal* ML model to perform user's specified tasks, whilst adapting to the complexity of IoT environments. In this section, we first discuss the model selection among three main categories i.e. TML, DL and

[2]https://github.com/Dash-Industry-Forum/dash.js

RL, followed by a survey of well-known models (or algorithms) in each category and their corresponding criteria for model selection.

*2.1.1 TML vs. DL vs. RL.* In this work we roughly divide the ML approaches/concepts into TML, DL and RL. Compared with the most popular DL , TML is relatively light-weight. It is a set of algorithms that directly transform the input data to certain feature spaces, according to certain criteria. For supervised cases when class label is available for training, TML aims to mapping the input data to the labels by optimising a model, which can be used to infer unseen data at the test stage. However, since the relationship between raw data and label might be highly non-linear, feature engineering— a heuristic trial-and-error process — is normally required to construct the appropriate input feature. The TML model is relatively simple, the interpretability (e.g., the relationship between the engineered features and the labels) tends to be high.

DL came into popularity in recent years. Consisting of multiple deep layers, DL is powerful to model complex non-linear relationship (between the model input and output) and thus does not require the aforementioned heuristic (and expensive) feature engineering process, making it a popular modelling approach in fields such as computer vision (CV) and natural language processing (NLP). DL models, however, tend to have more parameters to be estimated due to the complex structures. Hence it is normally data-hunger and may face overfitting problem especially when with small dataset. Meanwhile, DL's complex structures make it difficult to interpret the relationship between raw features and output, and accordingly inference time may be high when compared against TML models.

RL has become increasingly popular due to its success in addressing challenging sequential decision-making problems. Some of these achievements are based on the combination of DL and RL, i.e., Deep Reinforcement Learning. It has been shown considerable performance in IoT systems. In RL, there is usually one or more agents interacting with the outside environment, where optimal control policies are learnt through experience. Fig. 6 illustrates the iterative interaction circle, where the agent starts without knowing anything about environment or task. Each time the agent takes action based on the environment states, and it receives a reward from the environment. RL optimises this process such that it learns to make decisions with higher rewards received.

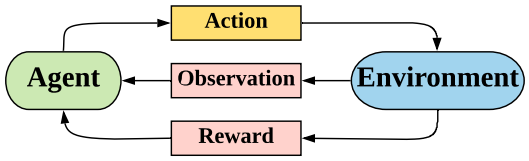


Fig. 6. Reinforcement Learning Paradigm

**Discussion.** In IoT environments, a variety of problems can be modelled by using the aforementioned three approaches. The applications range from system and networking [168] [167], smart city [310] [147], to smart grid [283] [223], etc. To begin with modelling, it is essential for users to choose a suitable learning concept at the first stage. The main selection criteria can be divided into two categories: *Function-based selection* and *Power Consumption-based selection*.

*Function-based selection* aims to choose an appropriate concept based on their functional difference. For example, RL benefits from its iterative environment ↔ agent interaction property, and can be applied to various applications which need interaction with environment or system such as smart temperature control system, or recommendation system (with cold start problem). On the other hand, TML algorithms are more suitable for modelling structured data (with high-level semantic attributes), especially when interpretability is required. DL models are typically used to model the complex unstructured data, e.g., images, audios, time-series data, etc. and it is an ideal choice especially when with high amount of training data and low requirement on interpretability.

*Power Consumption-based selection* aims to choose an appropriate model given constraints in computational power or latency. In contrast to TML, RL and DL are normally classified as high computational cost models as they usually contain complex network structures, and their accuracies tend to be more superior than the TML at a cost of high computational overhead. Although TML is normally more efficient at inference stage, reasonable accuracy can only be achieved with appropriate features (e.g., high level attributes derived from feature engineering).

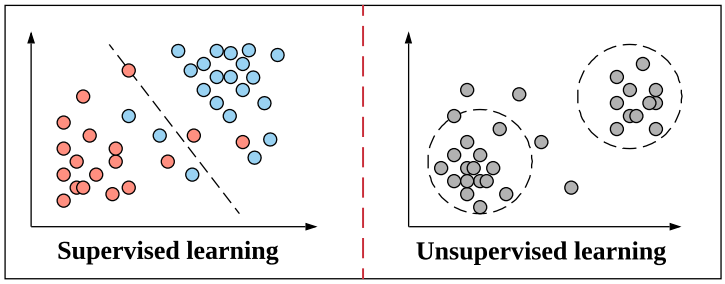


Fig. 7. Examples of Supervised Learning (Linear Regression) and Unsupervised Learning (Clustering)

*2.1.2 Traditional Machine Learning.* Herein we demonstrate several popular TML algorithms (Algorithm details can be referred to TML method in **Appendix B**), and discuss the criteria of choosing the TML algorithms. Given different tasks, TML can be further divided into *Supervised Learning* and *Unsupervised Learning*.

**Supervised Learning.** Supervised learning algorithm (i.e. Fig 7) can be used when both the input data $X$ and the corresponding labels $Y$ are provided (for training), and it aims to learn a mapping function such that $Y :\leftarrow f(X)$. Supervised learning algorithms have been widely used on IoT applications, hereby we introduce the most representative classifiers as below.

*Perceptron* and *Logistic Regression (LR)* are probably the simplest linear classifiers. For both models, the model (i.e., weights and bias) is basically a simple linear transformation. Perceptron can perform binary classification simply based on the sign of the (linearly) transformed input data, while LR will further scale the transformed value into probability (via *sigmoid* function), before a thresholding function is applied for the binary classification decision. LR can also be extended to process multi-class classification scenarios by using *softmax* as the scaling function, with class-wise probabilities as output.

*Artificial Neural Networks (ANN)* is a general extension of the aforementioned linear classifiers. Compared with *Perceptron* or *LR* which linearly project input data to the output, *ANN* has an additional "hidden layer" (with a non-linear activation function), which enables *ANN* to model non-linearity. However, in contrast to linear classifiers, this additional hidden layer makes it less obvious to see the relationship between the input and output data (i.e., low interpretability). Although it theory, with one hidden layer *ANN* can model any complex non-linear functions, in practise it has limited generalization capabilities when facing unseen data. *ANN* with more layers, also referred to as deep neural networks, tend to have better modelling capability, which will be introduced in the next subsection.

*Decision Tree (DT)* [205] and *Random Forest (RF)* [31] are two tree-structure based non-linear classifiers. Based on certain attribute-splitting criteria (e.g., *Information Gain* or *Gini Impurity*), *DT* can analyse the most informative attributes sequentially (i.e., splitting) until the final decision can be made. The tree structure makes it interpretable and it has reasonable accuracy when with low-dimensional semantic attributes. However, it faces "the curse of dimensionality" problem and doesn't generalise well when the input feature quality is low. *RF*, on the other hand, can effectively address this overfitting issue. *RF* is an ensemble approach on aggregating different small-scale DTs, which are derived based on

random sampling the features/datasets. The random sampling mechanism can effectively reduce the dimensionality (for each individual *DT*) while the aggregation function can smooth the uncertainty of individual *DT*s, making *RF* a powerful model with great generalisation capabilities. However, the interpretability of *RF* tends to be less obvious than *DT*, owing to the random sampling and aggregation mechanisms.

*Support Vector Machine (SVM)* [55] is another popular supervised learning method. It is also called large margin classifier as it aims at finding an hyperplane that is capable of separating the data points (belonging to different classes) with largest margin. For non-linearly separable datasets, various kernels (e.g., *RBF*) can be applied into the *SVM* framework with good generalisation ability. Yet the time complexity for training this algorithm can be very high (i.e. $O(N^3)$ [1], where $N$ represents the dataset size), making it less suitable for big datasets. On the other hand, *K-Nearest Neighbour (KNN)*[58], which does not require a training process (i.e., also referred to as lazy learning), is another powerful non-linear classifier. The classification is performed by distance calculation (between query and all the training examples), distance ranking, and majority voting among the ($K$) nearest neighbours. So selecting suitable distance functions/metrics (for different tasks) is one of the key issues in *KNN*. Since for any query sample, the distance calculation has to be performed for every sample in the whole training set, it can be time-consuming and thus less scalable to large datasets. Different from the aforementioned methods, *Naive Bayesian (NB)* algorithm [83] takes the prior knowledge of the class distribution into account. Based on the assumption that the features are conditionally independent, the likelihood of each feature can be calculated independently, before being combined with the prior probability according to the Bayes' rule. If the feature-independence assumption is not significantly violated (e.g., low-dimensional structured data), it can be a very effective and efficient tool.

**Unsupervised Learning.** The unsupervised learning algorithm (see Fig 7 right) aims at learning the inherent relationship between the data with only input data $X$ exists (without class label $Y$). For example, the clustering algorithm can be used to find the potential patterns of some unlabelled data and the obtained results can be used for future analysis. *K-Means*[105] and *Principal Component Analysis (PCA)* [237] are two most popular unsupervised learning algorithms. *K-means* aims to find $K$ group patterns from data by iteratively assigning each sample to different clusters based on the distance between the sample and the centroid of each cluster. *PCA* is normally used for dimensionality reduction, which can de-correlate the raw features before selecting the most informative ones.

**Discussion.** For IoT applications, a common principle is to select the algorithm with the highest performance in terms of effectiveness and efficiency. One can run all related algorithms (e.g., supervised, or unsupervised), before selecting the most appropriate one. For effectiveness, one has to define the most suitable evaluation metrics, which can be task-dependent, e.g., accuracy or mean-f1 score for classification tasks, or mean squared errors for regression, etc. Before model selection, a number of factors should be taken into account: data structure (structured data, or unstructured data which may need additional preprocessing), data size (small or large), prior knowledge (e.g., class distribution), data separability (linearly, or non-linearly separable which may require additional feature engineering), dimensionality (low, or high which may require dimensionality reduction), etc. There may also exist additional requirements from the users/stakeholders, e.g., interpretability for health diagnosis. Additionally, it is necessary to understand the efficiency requirement specific to an IoT application and one has to consider how the training/testing time grows with respect to data size. Time complexity shown in Table 2 in **Appendix B** provides more insights. Take *KNN* as an example: although no training time is taken, *KNN*'s inference time can be very high (especially when with large training set), and thus presumably unsuitable for certain time-critical IoT applications. Also, deployment environment is another non-negligible factor when developing IoT applications since many applications run (or partially) on low power computing resources.

*2.1.3 Deep Learning.* In this section, we primarily introduce three classical discriminant deep models (i.e., *Deep Neural Networks*/ *Multilayer Perceptron*, *Convolutional Neural Networks* and *Recurrent Neural Networks*) for supervised learning tasks on unstructured data such as image, video, text, time-series data, etc. We also brief two types of popular generative models: *Autoencoder*, and *Generative Adversarial Networks*.

**Deep Neural Network (DNN).** As previously mentioned, a deep neural network ( *DNN*) is a *ANN* with more than one hidden layers, and hence it is also called multilayer perceptron (*MLP*). Compared with *ANN* with a single hidden layer, *DNN* has more powerful modelling capabilities and its deep structure makes it much easier on learning the higher-level semantic features, which is crucial for classification tasks on complex data. However, for high-dimensional unstructured input data (such as images), there may be many model parameters to be estimated, and in this case, overfitting may occur if there are not enough labelled data. Nevertheless, generally *DNN* has decent performance when input dimensionality is not extremely high, and it has been successfully applied to various applications for example the human action recognition [266], traffic congestion prediction[66] and healthcare[186].

**Convolutional Neural Network (CNN).** When it comes to high-dimensional unstructured data such as images, in visual recognition tasks it is hard to directly map the raw image pixels into target label due to the complex non-linear relationship. The traditional way is to perform feature engineering, which is normally a trial-and-error process, and may require domain knowledge in certain circumstances, before TML is applied. This heuristic approach is normally time-consuming, and there exists substantial recognition errors even in simple tasks since it is very challenging to hand-engineer the high-level semantic features. *CNN*, a deep neural networks with convolutional layers and pooling layers, can address this issue effectively. The convolution operation can extract the higher level features while the pooling operation can keep the most informative responses and reduce the dimensionality. Compared with *DNN*, the weight sharing concept (of the convolution operation) enables *CNN* to capture the local pattern without suffering from the "curse of high-dimensionality" from the input. These operations and the hierarchical nature make *CNN* a powerful tool for extracting high-level semantic representations from raw image pixels directly, and successfully applied to various recognition tasks such as object recognition, image segmentation [80] and object detection [106]. Because of the decent performance on various visual analysis tasks, *CNN* is usually considered as the first choice for some camera-based IoT applications for example the traffic sign detection [240].

**Recurrent Neural Networks (RNN).** Nowadays, with the increasing amount of generated stream and sequential data from various sensors, time series analysis becomes popular among the machine learning (ML) community. *RNN* is a sequential modelling technique that can effectively combine the temporal information and current signal into the hidden units for time-series classification/prediction. An improved *RNN* named Long Short Term Memory (LSTM) [112], including complex gates and memory cell within the hidden units for "better memories", came into popularity in various applications such as speech recognition [95], video analysis [265], language translation [161], activity recognition [97]. etc. Since data streaming the most common form in the IoT environment, RNN(LSTM) is deemed as one of the most powerful modelling techniques, and there are various IoT applications such as smart assistant [82, 266], *smart car navigator system* [119], malware threat hunting [101], network traffic forecasting [208], equipment condition forecasting [304], Energy Demand Prediction system [182], load forecasting [131], etc.

**Generative Models.** *Autoencoder (AE)* [13] and *Generative Adversarial Network (GAN)* [90] are two popular generative DL models. Without requiring any label information, *AE* can extract compact features and reconstruct the original (high-dimensional) data with the extracted features. It is normally used for dimensionality reduction, latent distribution analysis or outlier detection. *GAN*, on the other hand, applies an adversarial process to learn the "real" distribution

from the input data. More precisely, *GAN* consists of two parts, namely generator and discriminator. The generator aims at generating indistinguishable samples compared to the real data. While the discriminator works adversarially to distinguish the generated fake samples from the real data. It is an iterative competition process that will eventually lead to a state where the generated samples are indistinguishable from the real data. With the learnt "real" distribution, one can generate various samples for different purposes. *AE* and *GAN* are both powerful tools in computer vision field, and their properties make them promising approaches for IoT applications. For *AE*, it can be used for diagnosis/fault detection tasks [49, 191] or simply as a preprocessing tool (i.e., feature extraction/dimensionality reduction). For *GAN*, it has been used for studies on generating rare category samples, and this upsampling approach may further improve the model performance [308, 309].

**Discussion.** The aforementioned DL models can be effective tools for processing different unstructured data types. The way of applying them is generally very flexible, and they can be used jointly to process the complex data from various sources in the IoT environments. For example, although *CNN*/ *RNN* could be used in an end-to-end manner (e.g., as image/time-series classifiers), they could also be used as feature extractors, based on which one can easily aggregate features extracted from different sources (e.g., audio, images, sensor data). When with high-dimensional video data, one can either model by training *CNN*+ *LSTM* jointly [271], or use *CNN*/ *AE* as feature extractors, before the sequential modelling (e.g., using *LSTM*). However, when modelling the data with limited label (e.g., rare event), one needs to consider the potential overfitting effect when using DL directly. One may go back to the TML approaches or use some upsampling techniques (e.g., *GAN*) to alleviate this effect.

*2.1.4 Reinforcement Learning (RL).* In this section, we first introduce the strategies used to formulate the aforementioned video streaming example (see §2) with Reinforcement Learning (RL). As mentioned earlier, in RL an *agent* interacts with the *environment*, learning an optimal control policy through experience. It requires three key elements, *observation*, *action*, and *reward*. Based on these, we can formulate the adaptive bitrate streaming problem. Specifically, *observation* can be the buffer occupancy, network throughput, etc. At each step, the *agent* decides the bitrate of the next chunk. A *reward* (for example the quality of service feedback from the user) is received after the *agent* takes *action* (chunk bitrate). The algorithm proposed in [168] collects and generalizes the results of performing the past decisions and optimizes its policy from different network conditions. This RL-based algorithm can also make the system robust to various environmental noises such as unseen network conditions, video properties, etc.

As shown in Fig 18, there are a plethora of algorithms in the whole reinforcement learning family. More details of these RL algorithms can be found in the **RL methods in Appendix B**, and here we focus on selecting appropriate RL algorithms based on different selecting criteria.

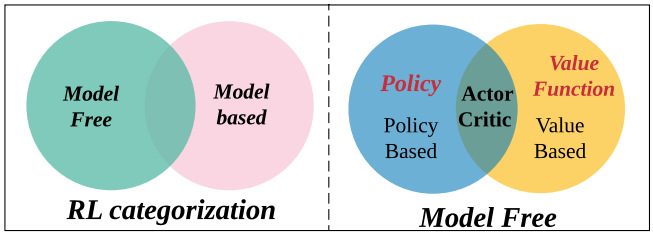


Fig. 8. Reinforcement Learning Categorization

**Environment Modelling Cost** In RL modelling, sample efficiency is one of major challenges. Normally the RL agent can interact either with the real world or a simulated environment during training. However, it can be difficult to simulate the heterogeneous IoT environments and complex IoT devices. RL models can also be trained directly in

real world IoT environments, yet one major limitation is the heavy training cost, which may range from seconds to minutes for each step. The model-based RL methods, a method that can reduce the sample complexity, can decrease the training time significantly. It first learns a predictive model of the real world, based on which the decisions can be made. When compared with model-free approaches, model-based methods are still in its infancy, and because of the efficiency property, they may attract more attentions in the near future.

**Action Space:** The action space of RL algorithms can be either continuous or discrete. For those RL algorithms with discrete action space, they choose from finite number of actions at runtime. Take the Video streaming task for example, the action space is different bitrates for each chunk. Another task formulated in discrete action space can be found in [167], where the action space is the "schedule of the job at $i$-th slot". Available algorithms for discrete action space tasks most reside in the policy gradient group, for example DQN, DDQN. The continuous action space, on the other hand, is infinite for all possible actions. Relationship exists between the actions that are usually sampled from certain distribution like Gaussian distribution. For example, in an energy-harvesting management system, PPO algorithm [231] is used to control IoT nodes for power allocation. The action space, as stated in [184] is sampled from a gaussian distribution to denote the load of each node ranging from 0% to 100%. Similarly, another work [7] that studied energy harvesting WSNs, Actor-Critic [130] algorithm is implemented to control the packet rate during transmission. One advantage of continuous action space lies in its ability to accurately control the system, thus a higher QoE is expected.

### 2.2 Model Generation

Based on the user requirement and task specification, we have selected a variety of models. Next, the models need to be developed and implemented. In this section, we will introduce the available tools for the development. We will also present the approaches that can be utilized to accelerate the training process.

*2.2.1 Machine Learning developing framework.* The training and execution of ML models can be tricky and it may require numerous engineering efforts. Efforts have been devoted to developing frameworks to support the model development. These frameworks have their own strength and weakness in terms of the supported models, usability, scalability, etc. In this section, we will review several development frameworks.

For TML, the most famous development framework is Sci-kit learn It is a free ML library with Python interface. Sci-kit learn supports almost all main-stream machine learning models and is a popular tool for fast prototype. For DL, we list some of the most popular DL frameworks and discuss their pros and cons in table 1. The users can choose the most suitable frameworks based on their needs.

Note that Tensorflow (TF) 2 eases the development of the DL networks with the eager execution and the TF ecosystem has made the whole deep learning development easy to be configured. This brings the convenience for users who want to develop DL models for various IoT devices. Further, TVM [44] breaks the boundaries among diverse hardware, which allows a DL model to be deployed and executed in many devices.

*2.2.2 Model acceleration.* Modern DL-IoT applications feature heterogeneous geographical distribution and large volume streaming data. The training of the DL models are increasingly complex due to the large data and computation volume as well as the complicated model design. It may require days or weeks to train a large model on a single machine. Moreover, on a single machine out of memory (OOM) problems may occur, making the training extremely hard or even impossible. High-performance computing clusters provide an opportunity to distribute the training across different nodes, greatly improve the training efficiency.

| DL frameworks | Core language | Interface | Pros | Cons |
|---|---|---|---|---|
| Tensorflow (2) | C++ | Python, Javascript, C++, Java, Go | - Effective data visualization<br>- Distributed learning<br>- Efficient model serving<br>- On-device inference with low latency for mobile devices<br>- Eager Execution with TF2, easy to debug | - Steep learning curve (migration from TF 1 to TF 2)<br>- Poor results for speed |
| Pytorch | C/C++ | Python, C++ | - Simple and transparent modeling<br>- Eager execution | - Hard to serve even with ONNX support |
| Caffe (2) | C++ | Python, C++ | - Fast, scalable, and lightweight<br>- Server optimized inference | - Limited community support<br>- Limited in implementing complex networks |
| Mxnet | C++ | Python, C++, Java, Julia, R, Perl, Closure | - Fast, flexible, and efficient in terms of running DL algorithms<br>- Run on any device<br>- Easy model serving<br>- Highly scalable | - Smaller community compared with Tensorflow or Pytorch |
| DL4J | Java | Java, Clojure, Kotlin | - Robust, flexible and effective<br>- Works with Apache Hadoop and Spark | - Robust, flexible and effective<br>- Works with Apache Hadoop and Spark |

Table 1. Comparison of Deep Learning Frameworks

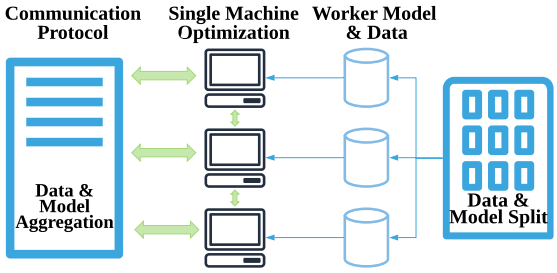


Fig. 9. Distributed Machine Learning Architecture

Fig 9 shows the operation pipeline of a typical distributed ML system. The data and models are first split and sent to a set of worker machines. On each machine, the models are optimized according to the hardware specifications. With proper network configuration, a central server will coordinate all worker machines during the training. The convergence of the training process can be greatly accelerated to support the iterative model generation experiments.

In this section, we discuss model acceleration in the context of distributed machine learning. First we review single machine optimization techniques. Then, we present several data and models parallelism strategies for better concurrency. Finally from a system perspective, we survey orchestrating mechanisms for distributed machine learning systems.

**Single Machine Concurrency.** The basic computation unit of neural networks consists of vector-vector, vector-matrix and matrix-matrix operations. Efficient implementation of computations can accelerate the training and inference process. The Basic Linear Algebra Subprogram (BLAS) standardizes the building blocks for basic vector, matrix operations. Higher level linear algebra library such as cuBLAS implements BLAS on top of NVIDIA CUDA and is efficient in utilizing the GPU computation resource. Intel Math Kernel Library (MKL) on the other hand, maximizes performance on Intel processors. The ease of use APIs optimize code and is compatible with BLAS without the change of code.

Different DL architectures (e.g., DNNs, CNNs and RNNs) may require different optimizations in terms of basic computations. The DNN computation is usually basic matrix-matrix multiplication and the aforementioned BLAS libraries can efficiently accelerate the computations with GPU resources. The CNNs and RNNs are different in their convolution and recurrent computations. Convolutions can not fully utilize the multi-core processors and the acceleration can be achieved by unrolling the convolution [41] to matrix-matrix computation or computing convolutions as point-wise product [170]. For RNN (LSTM), the complex gate structures and consecutive recurrent layers differ from the DNNs and CNNs, and no direct parallel can be applied. The optimization is possible though, with implementations on

top of NVIDIA cuDNN [46]. Computations among the same gates can be grouped into larger matrix operations [8] and save intermediate steps. We can also accelerate by caching RNN units weights with the GPU's inverted memory hierarchy [67]. The weights are reusable between time steps, making a maximum 30× speed up on a TitanX GPU.

Apart from the resource utilization optimization, the algorithmic level optimization methods can contribute to faster convergence. The learning of the ML model is performed by minimizing a loss function measuring the difference between the ground truth and model prediction. Various methods are iterative, such as BFGS [189], heuristics evolutionary approach [18], prominently performed Gradient Descent approach or Stochastic Gradient Descent approach [216]. In real training, however, modifications to the basic gradient descent algorithm assist in avoiding local optimum and thus faster convergence can be achieved. Most prevalent optimization algorithms are listed: Mini-batch Gradient Descent, ADAM [126], Momentum [202], AdaGrad [72], RMSProp [109].

**Training Concurrency in Distributed ML.** In the distributed machine learning, the selection of parallel strategies depends on the availability of hardware resources. Among the most frequently used strategies, here we introduce *data parallel* and *model parallel*.

*Data Parallel.* In a multi-core system where a single core can not store all the data, data parallel is considered with either splitting the data samples or the features. Data parallel has been successfully applied to machine learning algorithms [50] with each core working independently on a subset of data. One of the earliest work trained with GPUs can be found in [207] where the authors implemented distributed mini-batch SGD unsupervised learning concurrently with thousands of threads in a single GPU. Parallel with the split of data features, though, can not be used directly with neural networks because different dimensions of the features are highly correlated. However, it can be used when training other ML algorithm for example decision trees and other linear models where the features are relatively independent.

*Model Parallel.* When the size of the model is beyond the capability of a single node, they can be divided to sub-models and deployed to different nodes for parallel execution. The neural networks can be divided according to computation graphs horizontally or vertically. In DistBeilef [64], DNNs are divided into small parts and distributed over various nodes. Parameter update communication exists only between edges in the computation graph that across each other. As for the CNNs, modifications such as Local Connected Layers (LCN) in [140] can be engineered through the HPC infrastructure [52].

*Hybrid.* The data and model parallel are not mutually exclusive and the hybrid of both scheme is common in practice. With data parallel for CNN layers and model parallel for DNN layers, based on 8 GPUs the work in [133] achieved a 6.25× speed up with only 1% of accuracy loss. As a result, most of the large scalable ML training systems [47, 64] are designed based on these two strategies.

**Distributed ML System.** When we have acquired local model update with partial data slice, multi-node and multi-thread collaboration are important for effectively updating the model. Network communication plays an important role in sharing the information across the nodes. In this section, we present three most important features in network communication: 1) network topology, 2) update strategy, 3) communication frequency.

*Network Topology.* The network topology defines the nodes connection approach in the distributed machine learning system. When the data and models are relatively simple, it is common to utilize existing Message Passing Interface (MPI) or MapReduce infrastructure for the training. Later when the systems are becoming more and more complex, new topologies should be designed to facilitate the parameter update.

The Iterative MapReduce (IMR) or AllReduce approach are usually used for synchronous data parallel training. Typical IMR engines (for example the Spark MLlib [172]) generalizes MapReduce and enables iterative training required by most ML algorithms. Synchronous training can also be implemented by AllReduce with MPI. Libraries such as Gloo are leveraged to customize AllReduce in Caffe2 and DeepSpeech [6] implements Ring AllReduce. Both approaches are capable of utilizing multiple GPU nodes while keeping the communication overhead at a minimum.

A Parameter Server (PS) infrastructure [148] is beneficial for both performance and fault tolerance when part of the nodes broke down. It is usually composed of a set of worker nodes and a server node which gathers and distributes computation from the former ones. The asynchronous training neglects stragglers and is thus faster in convergence. Famous projects such as DMTK Microsoft Multiverso [74], Petuum [289] and DistBelief [64] enable training of even larger networks.

*Update Strategy.* There are two strategies to update the parameters across multiple nodes: synchronous and asynchronous.

Synchronous algorithms work by updating all the worker machines at the same time. A typical synchronous training algorithm is *Synchronous Stochastic Gradient Descent (SSGD)* [216]. SSGD is a less impractical method due to the time wasted on waiting for the stragglers in the worker nodes. This problem can be mitigated by introducing the backup workers [42] in the network. In a parameter server setting, $b$ backup workers in addition to the $N$ basic workers are used. Once the parameter server receives gradients from any $N$ worker nodes, the update of the global weights is triggered and the training proceeds to the next iteration. [260] studied that a carefully designed data replicas can reduce the impact of failures and stragglers for Synchronous Gradient Descent. In addition to the computation load and stragglers, [296] formalizes a three-fold trade-off: computation load, communication cost and straggler tolerance and achieves a 23% reduction on the running time.

The most popular asynchronous SGD (ASGD) algorithm is Hogwild! [214]. In this asynchronous setting, all worker nodes access the global model via a shared memory. They can pull and update global model parameters any time when the training is finished. This approach takes advantage of the sparse ML problems to avoid memory overwriting in a single machine multi-processor shared physical memory environment. Later in [192] a conflict-free Cyclades algorithm was presented to further increase the speeding while maintaining serial equivalence. The distributed version of ASGD can be seen in [190] which is scalable to multiple threads, GPUs and machines.

The aforementioned SSGD and ASGD are not optimal in real industrial setting. In order to speed with the correctness guaranteed, *staleness* [111] was introduced in compromise of both methods. In a stale-Synchronous version, the node pull updates from the server by a lagging of bounded $s$ time steps. With active check on stragglers, this approach performs well in heterogeneous environments [305].

*Communication Frequency.* Communication overhead is the key and often the bottleneck in distributed machine learning [149]. The sequential optimization algorithms implemented in the worker nodes require frequent read and write from the global shared parameters which poses great challenge on balancing network bandwidth and communication frequency. Smaller communication frequency can reduce the communication overhead where worker nodes train for more iterations before each communication. This approach balances the communication-computation ratio while requires more effort on the hyper-parameter tuning such as the local iteration steps and learning rates [256]. In an asynchronous setting, the communication frequency can also be maneuvered through the push and pull operations in the worker nodes. DistBelief [64] has adopted this approach with larger push interval compared to the pull interval.

*2.2.3 Discussion.* Effort has been devoted to implement the distributed machine learning on top of modern deep learning frameworks. Remarkable results have been achieved where with proper implementation [93] with Tensorflow,

the training time of the state-of-art ImageNet can be reduced from days to 1 hour. Compared with Tensorflow, more efficient implementation such as Horovod can increase the GPU utilization for even more acceleration. Horovod has already incorporated in various deep learning framework ecosystem. (e.g. Pytorch, Mxnet)

The deep learning ecosystems free the researcher from heavy implementation effort. There are however, challenges for model generation in a distributed setting: (1) *The choice of hardwares.* The same implementation can have different performance on different devices. One would have to be aware of the device features for efficient acceleration. (2) *Parallel hyperparameter tuning strategy.* Compared with single machine training, the distributed system is more complex and is thus more difficult to find an optimal structure. (3) *Effective work of DL frameworks with other big data application like Hadoop/Spark.* The existing big data framework (e.g. Spark/Hadoop) can also be applied for effectively distributing the DL training pipeline, and a deeper integration of both frameworks is urgently required.

### 2.3 Model Optimization

We have discussed the model selection and model generation where a model is generated catering to the specific needs of IoT applications. There are, however, still things to be considered before model deployment. The IoT application differs significantly from other areas in terms of deployment devices and data sources. The limited computational budget of edge devices requires smaller models for small-scale computational workload to ensure the low inference latency. Also, heterogeneous data sources in IoT environments usually contains redundant information that can even mislead the decision of the ML models. It is important to select only relevant and informative features or to perform model compression for performance optimization. In this section, we discuss these two topics.

*2.3.1 Feature Selection.* The amount of high-dimensional data in IoT environments pose challenges on the training of the ML algorithms. Noisy and redundant signals exist and they may consume substantial computational power. For unstructured data (e.g., images, audio, etc.), a common way to reduce the data redundancy is to perform feature extraction, which can be realised in many ways, e.g., via feature engineering, supervised/unsupervised learning, etc. The extracted features can be deemed as structured data, and its redundancy can be further reduced by feature selection. Feature selection can help reducing the computational complexity, which may improve the performance in terms of both effectiveness and efficiency—crucial factors in the limited-resource IoT environments. Briefly, feature selection is the process of preserving relevant features while discarding irrelevant/redundant features. There are generally three categories of feature selection approaches , namely the *Filter* approach, the *Wrapper* approach and the *Embedded* approach. In this section, we introduce each method and list several popular works within the category. After that, we survey the representative feature selection applications in IoT and propose future directions for feature selection.

**Filter Approach.** The filter approach selects a subset of the features without involving the evaluation of the learning algorithms. Based on the intrinsic properties of the data, this approach carries out the feature selection in a fast and efficient fashion. A typical filter method consists of two steps. First, features are ranked based on certain evaluation criteria, then the most important features are preserved for further processing. Another popular filter approach is *Subset search* method. It searches and evaluates the discrimination power of different feature subsets based on certain metrics, such as consistency measure [63], correlation measure [102] and the mutual information [14].

**Wrapper Approach.** The wrapper approach involves the learning algorithm in the feature selection process. It trains one model for each selected feature combination, and evaluate the importance of the feature subsets on validation sets. In wrapper approaches, the application of training/validation process makes it computationally expensive, when

compared with filter approaches. The wrapper methods can be subdivided into *Heuristic Search* and *Exhaustive Search* algorithms according to how the optimal subset is generated.

The *Heuristic Search* algorithms [99, 169] incrementally add features to an empty feature set until the objective (performance) threshold is reached. In each iteration, only the feature with the highest validation performance score is selected. The process is repeated until the required number of features is reached.

The *Exhaustive Search* [88, 124] algorithms enumerate all possible feature combinations for optimal subset selection. The exhaustive search process (for wrapper approach) is hardly used in practise since the corresponding training/validation operations make it computationally prohibited.

**Embedded Approach.** The embedded methods try to reduce the computation time taken up for searching optimal subsets as in wrapper methods. It is implemented with feature selection algorithms embedded in the training process. Compared to wrapper methods, subset search in embedded approach is much faster since it is guided by clear criteria.

The most typical embedded technique is the decision tree (DT) algorithm . As previously introduced in §2.1.2, the growth of DT is guided by the information gain or Gini impurity. Starting from a root node, the DT selects features and divides the sample set into small subset. The higher the information gain/Gini impurity, the more information the feature contains. The training of the algorithm and the selection of the feature is thus carried out simultaneously. ID3 [204], C4.5 [206], and CART [32] are among the most popular decision tree algorithms.

**Application and Future Application of feature selection in IoT.**

In *Smart Farming*, early detection and treatment of plant disease or nutrient deficiency would increase the health of the crops. An IoT based *Smart Farming* system can give real-time decisions with optimal fertilizer volume. [125] [174] show that when there are many sensors with large volume of IoT streaming data, feature selection can remove the redundant information and maximize the prediction accuracy. Another application is *Intrusion Detection*, which aims to detect online behaviour that compromises the confidentiality of the network. In this case, feature selection can be used to extract the most informative patterns [139, 143], serving as the basis of an effective intrusion detection system.

For future research, one could extend the single-object optimization to multi-object optimization. For example in IoT systems, optimal feature selection algorithms assist the machine learning models to optimize the execution time. We can explore the modifications of the feature selection algorithm to minimize the energy consumption of routing decisions as well [73]. Alternatively, one can study how to detect the dynamics within the data flow and then adaptively apply the search algorithms accordingly to further improve the performance of the feature selection algorithms.

*2.3.2 Model efficiency.* The state-of-art DL models often require high computational resources beyond the capabilities of IoT devices. Those models that perform well on large CPUs and GPUs cluster, may suffer from unacceptable inference latency or even unable to run on edge devices (e.g. Raspberry Pi). Tuning the deep neural network architectures to increase the efficiency without sacrificing much accuracy has been an active research area. In this section, we will cover three main optimization directions: *Efficient architecture design*, *Neural architecture search* and *Model compression*.

**Efficient architecture design.** There exist neural networks that can specifically match the resource and application constraints. They aim to explore highly efficient basic architecture specially designed for platforms such mobiles, robots as well as other IoT devices. MobileNets [114] is among the most famous work which proposed to use depth-wise separable convolutions [242] to build CNN models. By controlling the network hyper-parameters, MobileNets can strike an optimal balance between the accuracy and the constraints (e.g., computing resources). Later in MobileNetv2 [226],

the inverted residual with linear bottleneck architecture was introduced to significantly reduce the operations and memory usage. Other important works include Xception [48] ShuffleNet [307], ShuffleNetv2 [163], CondenseNet [117].

**Neural architecture search (NAS).** Another research direction named neural architecture search aims at searching an optimal network structure in a predefined search space. There are usually three types of algorithms: reinforcement learning approach [155, 200], Genetic Algorithm (GA) based [156, 213], and other algorithms [11, 34].

The models generated by these methods are normally constrained to smaller model sizes. Model size and operation quality are two most common metrics to be optimized, over other metrics such as inference time or power consumption. Representative works including MONAS [115], DPP-Net [69], RENA [316], Pareto-NASH [77] and MnasNet [259] are interested in finding the best model architectures to meet these constraints. These approaches are more straightforward as they optimize directly over real world performance.

**Model Compression.** As modern state-of-art DL models can be very large, in the case reducing the model computation cost is crucial for deploying the models on IoT devices, especially for those latency-sensitive real-time applications. Model compression methods can be divided into four categories, *Parameter pruning and sharing*, *Low-rank factorization*, *Transferred/compact convolutional filters* and *Knowledge distillation*. Hereby we briefly summaries the categories of model compression techniques and list several important works.

*Parameter pruning and sharing* method aims to find and remove the redundant parameters (of DL models) for higher efficiency and generalization. One direction is to apply quantization techniques. The DL's parameters/weights are usually stored in memory with 32-bits, and quantization techniques can compress them into 16-bits [98], 8-bits [269] or even 1-bit [56, 57, 212]. On the other hand, weight pruning and sharing (in pre-trained DL models) also raised interest among the community. Some popular methods imposed L1 or L2 regularization constraints [142, 281], which can penalize models with more parameters, yielding the effect of pruning unnecessary parameters.

*Low-rank factorization* method decomposes the CNN or DNN tensors to lower ranks. The tensor matrix decomposition is implemented for each layer of the DL model. That is, once the decomposition for the a certain layer is completed, the parameter size will be fixed (for this layer) and the decomposition will proceed to the next layer. Some interesting work can be found in [141, 141]. However, there are two major drawbacks. For very large DL models, it may be very expensive to perform decomposition owning to large parameter matrices. On the other hand, its layer-wise nature may also yield cumulative error, diverting the compression results far from optimal.

*Transferred/compact convolutional filters* method reduces the memory consumption by implementing special structural convolutional filters. Motivated by the equivariant group theory [53], the transferred convolutional filter transforms the network layers to a more compact structure, thus reducing the overall parameter space. The family of transformation functions [145, 235, 299] operates in the spatial domain of the convolutional filters to compress the whole network parameters. Compared to other model compression methods, transferred convolutional filters methods are less stable due to the strong transfer assumptions. However, when the assumption holds, the compact convolutional filter can have very good performance. In [258, 285], the filters were decomposed from $3 \times 3$ or bigger to $1 \times 1$ convolutions—ideal operations for the IoT devices.

*Knowledge distillation* method learns a new, more compact model that mimics the function presented by the original complex DL model. The idea came from the work in [35], where a neural network was applied to mimic the behaviour of a large classifier ensemble system. Later this idea has been extended to the complex DL methods [110], and more details can be found in [12, 219, 298]. However, currently the knowledge distillation methods are limited to classification tasks and further development is required.

### 2.4 Model Evaluation

After the models have been trained, based on suitable metrics their performance should be evaluated before deployment. Accuracy is one of the most popular evaluation metrics in classification tasks, yet it faces several problems in different scenarios. For example, it is an overall measure without indicating the recognition capability for each class, which may be heavily biased if there exists a significant class imbalanced problem. There are various evaluation metrics and it is key to select the most appropriate one. For the rest of this section we investigate several widely used metrics for classification and regression tasks. For classification/regression tasks, one aims to construct a model (i.e. $f(\cdot)$) that can predict the value of dependent variable $Y$ from independent variable $X$. The difference between these two tasks is the fact that the dependent variable $Y$ is numerical for regression and categorical for classification.

*2.4.1 Classification Problem based metric.* In classification tasks, one of the most effective evaluation metrics is confusion matrix [262]. As demonstrated in Table 2 for a binary classification task, in confusion matrix the row represents the predicted class and the column represents the ground truth (actual class). The entries True Positive ($tp$) and True Negative ($tn$) represent the correctly classified positive and negative samples, while the entries False Negative ($fn$) and False Positive ($fp$) denote the misclassified positive and negative samples, respectively.

| | Actual Positive Class | Actual Negative Class |
|---|---|---|
| Predicted Positive Class | True Positive ($tp$) | False Negative ($fn$) |
| Predicted Negative Class | False Positive ($fp$) | True Negative ($tn$) |

Table 2. Confusion Matrix for Classification

Based on confusion matrix, several evaluation metrics can be derived. The *accuracy* (i.e., $\frac{tp+tn}{tp+tn+fp+fn}$)and *error rate*(i.e., $\frac{fp+fn}{tp+tn+fp+fn}$ or $1-accuracy$) are the most commonly used metrics because it is more understandable and intuitive for human. However, these two metrics are powerless in terms of class-wise informativeness[164], which may neglect the minority class[40] (if there is a class imbalanced problem).

The metrics *precision* and *recall* can be used to measure the performance irrespective of the class imbalance problem (More definitions of classification evaluation metrics can be found in **Table 4 in Appendix C**). In binary classification problems, as mentioned earlier $tp$, $fn$, $fp$ are defined as the the number of "positives correctly classified as positives", "positives incorrectly classified as negatives" , "negatives incorrectly classified as positives" respectively. Then we can see *recall* (i.e., $\frac{tp}{tp+fn}$) indicates the ability of a classifier on detecting (true) positives out of all positive instances, while *precision* (i.e., $\frac{tp}{tp+fp}$) is the percentage of detected (true) positives out of all the detected ones. Since the binary classification's decision may highly depend on the threshold, there is a trade-off between precision and recall. For example, if a high threshold has been chosen—the similarity scores (the model outputs) have to be higher to give positive decisions—the classifier tends to have high $fn$ and low $fp$, yielding low recall and high precision. Similarly, reducing the value of threshold may increase recall and decrease precision accordingly. For different applications, one need to consider the optimal threshold for their requirements. For example, forensic applications may prioritize high precision (i.e., low in $fp$) while the medical diagnosis may prioritize high recall (i.e., low in $fn$).

In some tasks when both recall and precision are important, the *F1-score* (i.e., $2\frac{precision \cdot recall}{precision+recall}$)—a measure that can balance the precision/recall trade-off— is normally used. It is worth noting that for multi-class cases, the multi-class confusion matrix can be calculated, and the aforementioned precision/recall/F1-score can be extended to measure the class-wise performance. Depending on the data/applications, the overall performance can be measured by aggregating all the class-wise metrics. Two popular aggregation operations are averaging, and weighted averaging, e.g., mean F1-score, or weighted F1-score (over all the class-wise F1-scores).

*2.4.2 Regression Problem based metric.* For regression problems, the evaluation metrics are different from the classification ones. Popular evaluation metrics include Mean Squared Error (MSE), Mean Absolute Error (MAE), Mean Percentage Error (MPE), etc. Details and formulas of these metrics can be found in **Table 5 Appendix C**.

*Mean Absolute Error (MAE)* and *Mean Squared Error (MSE)* are the simplest metrics for regression evaluation. They denote the expected model errors defined in terms of absolute difference and squared difference (between the predicted value and the ground truth), respectively. Alternatively, the *Mean Absolute Percentage Error (MAPE)* and *Mean Percentage Error (MAE)* can also be applied to regression problems. The MAPE is similar to MAE but more intuitive as it shows percentage. The MPE lacks the absolute term on MAPE, which means the positive and negative errors will cancel out. In this case, the MPE can not be directly used to measure the performance of model. However, it can be used to check if the model systematically underestimates (more negative error) or overestimates (more positive error).

All of the above metrics can be applied to the regression problem, but it's important to consider the property of the dataset beforehand. For example, some fields may (or may not) be more prone to outliers, and the corresponding (effective) evaluation metrics may be different.

## 3 MODEL DEPLOYMENT

When the ML models mentioned in the previous section are developed and may need to be deployed and composed as an application in the complex IoT environments. To simply the deployment, the ML models and underlying infrastructure need to be specified (§3.1). Next, the optimization techniques can be applied to generate the deployment plans that select the suitable ML models for the deployment, optimizes the resources utilization of the model deployment, improves the reusability of the deployed models (§3.2). Once the deployment plans are generated, the models will be deployed over the specified infrastructure and the deployed models will be composed as defined in the plan (§3.3).

### 3.1 Declarative Machine Learning and Deployment

**Declarative ML.** Declarative ML aims to use high-level language to specify ML tasks by separating the applications from the underlying data representation, model training and computing resources. There are *three* general properties of declarative ML. First, the high-level specification only considers data types of input, intermediate results and output. They are exposed as abstract data types without considering the physical representation of the data or how the data is processed by the underlying ML models. Second, the ML tasks are specified as high-level operations through well-defined semantics. The basic operation primitives and their expected accuracy levels (or confidence interval) are defined accordingly. Based on the operation semantics, declarative ML systems select the features and underlying ML models automatically or semi-automatically, optimize the model performance and accuracy for varying data characteristics and runtime environments. Notably, the selection is based on the available models, provided as services. Finally, the correctness of the ML models must be satisfied when a given model produces the equivalent results in any computing resources with the same input data and configurations. As a result, the declarative ML enables execution of the ML models over various hardwares and computation platforms (such as Apache Spark) without any changes.

SystemML [27] is an implementation of declarative ML on Apache Spark. Through domain-specific languages, it specifies the ML models as abstract data types and operations, independent of their implementation. The system is able to specify the majority of ML models: matrix factorizations, dimensions reduction, survival models for training and scoring, classification, descriptive statistics, clustering and regression. There are also other state-of-the-art research on declarative ML, including TUPAQ [248], Columbus [301]. They utilize language specification and modelling technologies

to describe the ML models for automatic model and feature selection, performance and resource optimization, model and data reuse.

**Declarative Deployment.** Hardwares in IoT environment consist of three basic type of devices: *data generating* devices, *data processing* devices and *data transferring* devices. *Data generating* devices are also called "Things" (e.g. sensors, CCTV) and are used to collect environmental data. *Data transferring* devices such as router, IoT gateway, base station are used to transfer the generated data to the *data processing* devices. *Data processing* devices are used to run the analytic jobs. They can be GPU, CPU and TPU servers running in cloud or ARM based edge device such as Raspberry Pi and Arduino. The ML-based IoT application is usually running across a fully distributed environment, such that it requires correct specification of the component devices as well as the precise interoperation between these devices. [244] lists fundamental aspects that may simplify the hardware specification, i.e., processor, clock rate, general purpose input/output (GPIO), connectivity methods (Wi-Fi, Bluetooth, wired connection) and communication protocols (serial peripheral interface), universal asynchronous receiver-transmitter (UART).

Regarding to the software, it is often categorized into three groups based on operation levels: *operating system (OS)*, *programming language* and *platform*. IoT *OS* allows users to achieve the basic behavior of a computer within internet-connected devices. The choice of OS in different layers of IoT environment depends on the hardware properties such as memory and cpu. The *programming language* helps the developers to build various applications in different working environments with diverse constraints. The choice depends on the capability of devices and the purpose of the application [36]. The IoT software *platform* is a system which simplifies the development and deployment of the ML-based IoT application. It is an essential element of a huge IoT ecosystem which can be leveraged to connect new elements to the system. For more details of the most popular OSs, programming languages and platforms in IoT domain, one can refer to **Appendix A**. The ML development *platforms* have been discussed in section 2.2.1.

The heterogeneity of IoT infrastructures makes the deployment very complicate and difficult to be automated. To overcome this issue, the infrastructure must be described and specified by machine understandable languages. Then, the declarative deployment systems are able to automatically map the ML models to the infrastructures and generate the deployment plans that optimizes the performance and the accuracy.

The declarative TOSCA model [61] is able to specify the common infrastructures such as Raspberry Pis and cloud VM (hardware), MQTT and XMPP (communication protocol). The deployment logic can be defined through *TOSCA Lifecycle Interface* that allows the users to customize the deployment steps. However, this declarative model is still very basic which can not handle the complex deployments such as specifying the details of ML based application. Moreover, the IoT applications consist of installing devices and sensors which require *human tasks*. These tasks are not natively supported by any available declarative deployment [33]. The imperative tool (e.g., kubectl commands) allows the technical expertises with diverse knowledge of different deployment systems and APIs to interact with a deployment system and decide what actions should be done. However, current imperative frameworks such as Juju, Kubernetes still do not support the interaction like sensor installation. In Future, the declarative deployment systems should interact with declarative ML systems to deploy a complex application over the heterogeneity of IoT infrastructure while supporting the *human* tasks through a more human centered imperative deployment model.

### 3.2 Deployment Optimization

When the infrastructures and deployment workflow of the ML models are specified, the deployment optimization problem can be formed as a mathematical expression subject to a set of system constraints. Then, resource allocation

algorithms can be used to efficiently and precisely find the best solution for the given mathematical expressions. Moreover, the optimization objectives are a set of QoS parameters including storage and memory space, budget, task execution time and communication delay etc,. These algorithms can be divided into *two* classes based on whether optimal solution can be guaranteed: *meta-heuristic method* and *iterative method* (or *mathematical optimization*). Nowadays, ML methods are getting popular and applied to solve these resource allocation problems by learning "good" solutions from the data. We investigate the representative works in resource allocation based on these *three* classes.

**Iterative-based method.** This class of algorithms generate a sequence of improved approximate solutions where each solution is driven by previous solutions. Eventually, the solutions will converge to an optimal point proofed by a rigorous mathematical analysis. The heuristic-based iterative methods are also very common, but we categorize this type of algorithms into the *meta-heuristic based method*. The most popular algorithms of this class include newton's method [165, 166], gradient method [15] and ellipsoid method [160]. To apply and adapt iterative-based algorithms to optimize resource allocation requires strong mathematical background, which can be an obstruction for software developers to utilize these algorithms to optimize their deployment. Furthermore, the algorithms have the variety of performance for different problems in terms of efficiency and accuracy. As a result, more algorithms from iterative-based method need to be studied and simplified by the system researchers, providing toolkits (or solvers) to tackle different optimization problems in IoT application deployment.

**Meta-heuristic based method.** The optimization problems in IoT applications can have large search space or are time-sensitive. The *meta-heuristic* based method is faster than *iterative-based method* in finding a near-optimal solution. This type of method consists of two subclasses: *trajectory*-based method and *population*-based method. The *trajectory*-based method finds a suitable solution with a trajectory defined in the search space. First, the resource allocation problems are mapped into a set of search problems such as variable neighborhood search, iterated local search, simulated annealing and tabu search. Then, the *meta-heuristic* algorithms are used to find the solutions. Many survey papers [104, 157, 245] have reviewed the algorithms applied for resource allocation in IoT, cloud computing, mobile computing. Additionally, the *population*-based methods aim to find a suitable solution in the search space that is described as the evolution of a population of solutions. This method is also called evolutionary computation and the most well-known algorithm is genetic algorithm. [300] investigates the resource allocation problems solved by evolutionary approaches in cloud computing.

**Machine learning based method.** ML based method is inspired by the abilities of data in representing the performance and utilization of the contemporary systems. The ML based methods are used to build data-driven models that allows the target systems to learning and generate an optimized deployment plan. The proposed algorithms have been used to optimize various QoS parameters such as latency [168, 293], resource utilization [175], energy consumption [19] and many others. Zhang etc., [302] have given a comprehensive survey of the ML based methods used for resource allocation in mobile and wireless networking.

Deployment (or resource allocation) optimization problems have been studied for decades, which remains a huge legacy for overcoming the optimization problems in deploying ML-based IoT applications. Instead of developing new optimization algorithms, more efforts are required to model the complex optimization problems, in which the system scale, conditions and diversity have been amplified significantly.

### 3.3 Model and Action Composition

Deployment of ML models in a pipeline requires proper model composition to maximize the user QoS. As shown in §1.1, a smart car navigator system comprises multiple ML models, including speech recognition, text classification, text generation and text-to-speech (TTS) model.

*Action composition* is defined by composing a set of basic actions for complex decisions. In a self-driving car operating system, actions can be accelerating, braking, turning left and right, etc. The combination of various action spaces increases the difficulties of learning optimal decisions in such complex systems. The Hierarchical abstract machines (HAM) [246] are well studied in the context of reinforcement learning [196, 261] by allowing agents to select from a constrained list of action spaces, speeding up the learning and adaptation to the new environment.

Model composition allows reuse of pre-trained models across different tasks, which substantially accelerates the development process. It can be formulated as a directed acyclic graph (DAG) with vertices representing the model and edges representing the data flow. Once the system receives a user query, it will deliver the query through all nodes and edges for process. The system will generate one response until the end of DAG. The modularity and flexibility of model and action composition simplify the development of ML-based IoT systems. We discuss how the configurations regarding the *model batch size*, *model replica* and *system buffering* can affect the performance of the whole system.

*Per-Model Batch Size.* Batching the received user queries optimizes throughput by fully utilizing the features of the pre-trained models, which is faster than processing one query at a time. However, batching query can potentially increase latency because the the model will wait for a whole batch of queries to come before it starts to proceed. The first query is not returned until the final query is processed [59]. The choice of the per-model batch is challenging due to the sequential composition between the models.

*Model Replica.* In heavy or bursting loads, system must quickly respond to the query fluctuations to meet the latency requirements. To alleviate the system congestion and achieve high throughput, it is critical to identify the bottleneck, which can be challenging due to the system dynamics. The bottleneck models can be resolved by replicating the model instances across multiple devices [60], therefore balancing the workload. However, distributing the queries across more model replica in a parallel setting [60] is challenging as well since the optimal placement depends on the model performance and the device capacity.

*System Buffering*. Serving system as a stream processing system comprises components across multiple devices. These devices usually process at different speeds, making system buffering across nodes necessary. Message queues are usually implemented to ensure smooth running within the system. However, buffering mechanism would increase the latency based on various system configuration [60]. It is thus challenging to design proper strategies to balance the the message queue overhead and the system latency.

## 4 MODEL AUDIT

Audit aims to evaluate whether the application is operating effectively, safely and reliably with the collected evidence. To this end, we must know what we should audit. Most of the works focus on monitoring or debugging the issues caused by infrastructure failures [135], implementation bugs [70, 132] and deployment errors [255]. In this section, we investigate the security, reliability and performance issues caused by ML models, especially DL models.

### 4.1 Security

There are many surveys regaeding IoT security issues and challenges. The security of IoT standardized communication protocols were evaluated in [94] based on their proposed model. [241] categorized the security issues of IoT into *eight* domains including authentication, access control, confidentiality, privacy, trust, secure middleware, mobile security and policy enforcement. [218] studied the main challenges and solutions of designing and deploying security mechanisms in centralized and distributed IoT architectures. [153] discussed the security features of IoT and categorized the attacks into *four* layers. i.e., the perception layer, the network layer, and the application layer.

In this subsection, we discuss security issues for deep learning based IoT applications: *Model exploratory attack*, *Data poisoning attacks* and *Evasion attacks*. The *model exploratory attacks* do not happen during the training. Instead the attacker tries to discover information from the trained model including the model itself and training data. It consists two types of attacks: *model stealing* and *membership leakage*. The *data poisoning attacks* happen during the training phase, where the attackers attempt to shift the boundary of DL models in their favor by polluting the training data. Finally, the *evasion attacks* maliciously craft the inputs for the deep learning based IoT application to trigger abnormal model behavior. Interestingly, the development of the research on adversarial learning has started the arms race between adversaries and defenders. The following subsection summarizes the most popular attacks with The defenses of these attacks explained in **Appendix D**. We also propose research directions for development of robust IoT applications.

*4.1.1 Model exploratory attack.* This type of attacks are usually performed on open-source frameworks such as PredictionIO and cloud-based machine learning service. This ML-as-a-service may allow users to input partial feature vectors while still able to receive confidence values in addition to prediction results. Thus, the attacker can leverage this feature to either extract the model or the sensitive information underlying the model. The *model stealing* and *membership leakage* are *two* main attacks of the model exploratory attack. The *model stealing* attack aims duplicate the functionality of the model, thereby evading detection of the model [9]. [264] proposed a method that learns the target models via a prediction API. Evaluations show that this method successfully extracts the models including logistic regression, SVM, neural network and decision tree from BigML and Amazon Web Service. More attack methods can be crated based on the extensive literature on learning theory, e.g., PAC learning [267] and its variants [16]. The *membership leakage* attacks are interested to steal the information from the training data which may not publicly available and may contain some sensitive information such as trade secrets, medical records etc. In this type of attack, an attacker is able to infer the members of the population or the members of the training dataset. Attacking the members of the population means that the types of data are used to create the model. Therefore, the target model has not generalization for the adversary, because he/she has the samples of the entire population of the training dataset. The attacks were successfully performed in the dataset including voice, handwritten images, network traffic, online shopping, record of hospital stays etc [9, 239]. The members of the training dataset attack aims to identify the individuals whose data are used for training a model, which cause a serious privacy issue. For example, if an attack knows that a patient's medical record was used to train a disease detection model, it also reveals that the patient has this disease. The experiments in [264] show that the attacks are able to extract the training dataset when the model is based on kernel logistic regression.

*4.1.2 Data poisoning attack.* Unlike *model exploratory attack*, an adversary performs the attacks during model training phase. These attacks insert carefully constructed poison instances into the training dataset to manipulate the performance of a system. The categories of data poisoning attacks are based on their targeting machine learning models i.e., supervised learning, unsupervised learning and deep learning.

*Data poisoning attack in supervised learning.* A causative attack was proposed by Xiao et al to against SVMs which utilizes the *label flipping* to poison the training data [288]. The *label flipping* attack attempts to add the noise label to the training data. Thus, these flipping labels are able to cause some malicious samples to be labeled as legal, or make the legal samples to be labeled as malicious. To improve the efficiency of the attack, Biggio and Laskov [21] utilized the gradient descent algorithm to find the best attack points for flip the labels.

*Data poisoning attack in unsupervised learning.* The poisoning attack has been demonstrated against various clustering algorithms. The idea is to introduce the carefully crafted data points to the training dataset to cause clusters to merge. In [22, 215], the authors assumed that the attacker has the full knowledge of the clustering algorithm and then reduced the attack to an optimization problem. The evaluations show that the clustering algorithms are compromised significantly with a very small percentage of poisoned input data.

*Data poisoning attack in deep learning.* There are very few data poisoning attacks in neural network. [250] showed that a deep learning model loss 11% accuracy after modifying 3% training data. Moreover, if the attacks are focus on attacking the specific test instances, the successful rate, time consumption and required resources (the number of modified samples) can be reduced significantly [45, 96, 250]. [233, 254] targeted to real-world scenarios where the labels are examined by human reviewers and malware detectors. The authors aimed to overfitting the deep learning models by poisoning the training data. Thus, the target instants (trained models) wouldn't performance well during inference time.

*4.1.3 Evasion attack.* With explosive development of machine learning, evasion attacks are becoming the most prevalent type of attack in machine learning, attracting people's eyes from both academic and industry. Fig 10 shows the arms race between the attacker and defender. It shows that the attacker attempts to confuse the defender with a crafted adversarial example, while the defender aims to strengthen its ability to filter out illegitimate input.

During both training and inference, the attacker can generate *adversarial examples* by modifying the samples. The training phase modification is similar to *data poisoning attacks* in that the decision boundary of the defender classifier is modified by insertion, modification or deletion of the training dataset. There are two approaches for generating adversarial samples, *white-box* or *black-box*. In *white-box* setup, the adversarial samples are crafted based on the attacker who has access to both the training data and the targeted model. Therefore, an adversary is able to obtain the boundaries of the targeted model by carefully modifying the training data. To be more explicit, as shown in fig 10, the defender aims to stop using the illegitimate input $X$ to train the itself. In order to fool the defender, the attacker attempts to learn the boundaries of defender by adding the perturbations to $X$ and then performing the attack. This process is repeated until the adversarial samples break the boundaries. The most representative techniques [91, 137, 194] are based on the attacker who has knowledge of both target model and instance of data. In the *black-box* setup, the attack introduced in [193] is not aware of the training data and the targeted model. The only observation of the targeted model is the inputs and their labels given by the targeted model. Based on this, a local model is trained to replace for the target DNN. The 84% adversarial examples generated by the local substitute model are misclassified by the targeted DNN.

*4.1.4 The system challenges of building a secure ML-based IoT application.* Most of the attacks and defenses reviewed in previous sections focus on developing the algorithms for *functional tasks* such as computer version, natural language processing, audio speech processing etc. To build a secure IoT application, we must consider the security issues from the system perspective, as the *functional tasks* cannot perform well when the system is under attack, We discuss two system challenges to improve the security of ML-based IoT application.

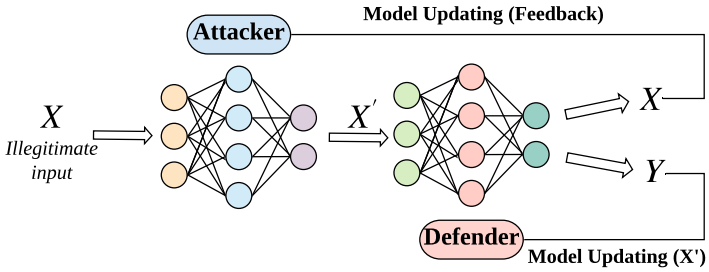


Fig. 10. Arms race game between the attacker and the defender. The attacker takes an illegitimate example ($X$) as the inputs of the his/her neural network and generate an adversarial example ($X'$). This example attempts to fool the defender which is a classification neural network. If the classifier recognizes the adversarial example as $Y$ which belongs to the legitimate input, it means the attacker wins the game. On the contrary, the defender wins the game if the adversarial example is classified as $X$. When the attacker fail, will try to update its model to generate a stronger adversary example based on the feedback. Similarly, the defender will enhance its model based on the lesson learned from the successful attack.

**Developing new attacking and defending models.** ML has been widely used in IoT system developments including network engineering [150, 168], resource allocation [146, 292], system debugging [38, 243], network intrusion detection [129, 158, 274] and network operations[116]. These systems can be exposed to the aforementioned attacks as well. Literatures [2, 78, 103] have revealed successful examples of attacks and defenses. Further the efforts on *functional tasks*, more research and development (R&D) work is required to improve the security of ML-based IoT applications.

**Developing new security platforms/frameworks.** We have discussed the arms race game between the attacker and defender (see Fig 10). This can be utilized to ensure the resilience of the IoT applications to various attacks. At a high level, an ideal platform would be able to launch various attacks via a predefined deployment pipeline to attack the experimental group. Meanwhile, the attack behaviors and system performance will be monitored to reinforce the capacity of the defender. To this end, three research questions need to be answered. 1) *How to automate the attacks?* Unlike the traditional software deployment problem, deploying attacks are much more complicated. For example in evasion attack, the proposed platform must be able to use various ML models to craft the adversarial examples. It is very difficult to automate this process. Due to the difference between the model inputs and outputs, the models may need to be retrained based on the observation of the real world to generate better adversarial examples. 2) *How to monitor the attacks?* As we discussed in previous sections, the attacks can be performed in data collection, model training and model inference. Therefore, the traditional log system is not able to handle this complexity. In data poisoning attack, for instance, the log system fails to capture the impact that the fake data points are injected to train a defender. 3) *How to coordinate the attacker with the defender?* At the high level, the arms race game between the attacker and the defender is very logical. The challenge here is to continuously select the suitable attacks thereby improve the defender performance. This can be formalized as an optimization problem where one of the objective is to maximize the ability of a system in defending certain type of attack.

### 4.2 Fault Tolerance

Distributed system fault tolerance has been studied for decades, many representative works have been proposed to handle the failures including system architecture [209] and algorithm design [37]. In IoT environment, the probability of failure increases significantly, and many faults are very hard to detect. Our previous paper [87, 284] reviewed the state-of-the-art research, and then discussed the key research directions. In this subsection, we aim to investigate the most common fault in ML applications, namely, lack of generalization.

Generalization ability (i.e., out-of-sample error[29]) measures the prediction accuracy of models for data samples that are not on the training dataset or from other distributions. A model with low generalization ability can be defined as the lack of generalization. It is a typical failure for deep learning models due to the stochastic optimization methods (i.e., wrong global minima [187]) or insufficient data samples. These lack of generalization issues are amplified in IoT application due to the dynamics and uncertainty found in the IoT environments. For example, car autopilot system causes car crash due to the failure of object detection algorithm in detecting the front cars under strong sunlight. Works attempted increase the generalization ability of ML algorithms which can be identified as *three* types: *Explicit regularization approach*, *Implicit regularization approach*, and *Data Augmentation Approach*.

*Explicit Regularization Approach.* A theoretical justification for regularization is applied on the solution *Occam's razor* [25]. In other words, regularization is a technique that discourages learning a more complex or flexible model, so as to avoid the risk of overfitting and to improve the generalization ability. As mentioned in §2.3.1, *Lasso* and *Ridge regression* are two *weight decay* [134] approaches that penalize the model parameters to prevent model from becoming too complex and incurring overfitting. *Dropout* [249] is one of the most interesting type of regularization techniques. During the training, the dropout approach randomly drops a fraction of nodes and connections in each iteration, which incurs different networks for each step. So that shallow networks can generalize better to different data distributions.

*Implicit Regularization Approach.* The implicit regularization approach penalizes the model parameters in an indirect way [121]. More precisely, the primary goal of these approaches are not to improve the generalization ability directly. For example, some methods are designed to solve the gradient vanishing/exploding problem or to improve the convergence speed. However, the empirical evidence shows that those implicit regularization approaches indeed improve the generalization ability of models. *Stochastic gradient descent (SGD)* [28] integrates a stochastic differential equation that prevents the model from stuck in a sharp minima [247]. *Batch Normalization (BN)* [118] aims to normalize the input of each layer to zero mean and unit variance, which stabilizes the gradient flow across different dimensions. Specifically, the smoothening effect of model reparametrization encourages faster converge to flat minima [228]. *Early stopping* [17] is another implicit regularization technique prevents the model from overfitting. It automatically stops training when the model generalization error continues to upgrade for certain iterations.

*Data Augmentation Approach.* The low generalization issue can also be caused by insufficient data samples as the deep learning models are prone to overfit with small data samples. In this case, generating valid data samples to increase the model generalization ability has attracted a lot of attention in the ML community. *Data Warping* generates the data samples by certain random process such as transformation. The traditional transformations achieve decent performance on various tasks such as cropping, rotating, and flipping [275]. Recently, benefit from the generative adversarial model (GAN), other approaches (i.e., *Data Generating*) have been used for data augmentation. Such approaches generate realistic data from real data distribution that are not shown on the original dataset [273]. Alternatively, adversarial samples can also be used to increase the generalization ability of the model [287].

### 4.3 Performance Evaluation

In this section, we consider several performance criteria that need to be considered for evaluating the efficiency of the obtained ML models. The criteria is identified as *two* main dimensions *model precision* and *execution latency*.

**Model Precision.** In a typical IoT application, the software performance is assumed stable after deployment. However, this is not the case for ML application where precision degradation is always expected after deployment. Precision

degradation can happen for various unexpected external changes that lead to shift in data distribution. The device location change, time and the weather are all important factors that may decrease the model performance. Therefore, it is critical that the model performance is monitored and new data is introduced continuously for retraining of the model. In ML, we define lifelong learning [195] as continually acquiring data, extracting new information without catastrophic forgetting of past knowledge. Lifelong learning keeps the model precision at a steady level.

**Execution Latency.** Many IoT applications are latency-sensitive depending on their tasks. For example, in the aforementioned smart transportation system (see §1.1) where sensors monitor and detect the car accidents, instant decisions have to be made to warn the drivers of potential hazards. Various factors, listed below, have to be evaluated to ensure seamless communication among the distributed components of a smart IoT application.

**Bandwidth Usage.** In distributed IoT networks, large scale IoT sensors are generating huge amount of data all the time. It is not possible to send all the data to cloud for data analysis. Fog computing proposed to move the computing close to the sensors to reduce the data transmission between the IoT network. However, the bandwidth of sensor network and edge network are still limited, some nodes may experiences high latency due to the network congestion. This may cause hug latency for the whole system as well. We need to monitor and evaluate this network dynamics [167] in order to provide solutions to alleviate the congestion in the networks.

**Resource Consumption.** Hardwares in IoT applications vary in computing power, memory and storage capacity. For any resource-intensive tasks for example those computation-heavy or memory-heavy ones, resource exhaustion in one node may lead to unacceptable latency for the whole application. It is thus necessary to design efficient resource management system [180, 297] to monitor and optimize task allocation for these physical devices,

**System Throughput.** The ML-based IoT applications may be developed to serve million's people, for example, the *smart traffic routing application* mentioned in 1.1. These massive number of users may send the requests simultaneously. Responding these requests quickly without losing users' satisfaction is still an unsolved problem in cloud computing. However, this issue is amplified in ML-based IoT applications, in which the queries may be performed on various devices and models. Some database optimization techniques such as caching frequent queries, batching queries and approximate computing are applied [60, 199]. There are still reminding the gaps in optimizing the query plans by considering heterogeneousness of the computing resources, uncertainty of the network, and diversity of ML models.

## 5 DATA ACQUISITION

Data is one of the most important constituents in developing a ML model as the prediction accuracy of the model has a strong positive correlation with the quality of the input data [176]. To provide high quality data for a ML-based IoT application, we orchestrate the data acquisition process into several steps. In the whole data acquisition pipeline, we collect raw data from various data sources (§5.1). With proper preprocessing techniques (§5.2) to removes redundant information and annotate the data, we are capable of performing several different ML tasks. While we have more data sources during the development process, we can also fuse (§5.3) them to provide more consistent and useful information. The following subsections will focus on the mentioned steps and discuss how data acquisition can support for developing a robust ML-based IoT application.

### 5.1 Data Collection

The IoT data can be broadly categorized into *Structured data* and *Unstructured data* based on its representation. *Structured data* can be represented in a pre-defined format (rows and columns). The meaning of each field is explicit which eases

the analysis and storage of the data. Examples of structured data include employee register information, visiting logs, etc. On the other hand, *Unstructured data* lacks any specific structure or format. Varying from text, audio, video to mails and messages, it accounts for a large proportion of IoT data. These two types of data are generated in *three* formats including *signal data*, *log data* and *packet data*. The *signal data* collects the daily life signal through various hardware such as sensors, sound recorders, cctv cameras, etc. The *log data* is usually used to capture the system status. Finally, the *packet data* is the data sent over the network and each unit transmitted consists of a header and the actual data. To collect these data, *two* important factors need to be considered: 1) *Data exchange*, 2) *Resource consumption*.

**Data exchange.** The data generated from a IoT device and send the data to an edge (sink) node or through other IoT devices and eventually the data will be collected and stored in the cloud. Also, the computation power of gateways and edge nodes are improving, which bring an opportunity to remove the data redundancy while saving the energy and bandwidths required for transferring data to downstream nodes [282]. This aggregation requires to apply various data summarization techniques [54] including sampling, sketching, histograms, wavelets and adapt these techniques to meet the constraints of the hardware and the time-varying channel conditions. Henriette etc., [217] investigated the state-of-the-arts stream process systems that can be used to implement the these data summarization techniques and execute them in parallel and elastic manner. However, it still requires a lot efforts to develop new data summarization techniques and stream process systems to handle the difficulty of processing the high volume data from various sources with multimodal.

**Resource consumption.** As aforementioned, IoT devices are very limited by resources such as available energy, wireless bandwidth, processing capability, storage capacity and battery power. Thus, it is very critical to optimize the resource utilization while transferring data from the devices to the cloud. To this end, we need to consider *three* issues: *resource allocation*, *energy control* and *task allocation*. Resource allocation in the context of data collection is to assign the computing resources, e.g., time slot, communication bandwidth, CPU and memory, to IoT devices for performing their tasks. Different with other resources, the energy may be provided by battery which limits the service life of the devices. Energy control focuses on optimizing the energy consumption when the IoT data is processed and transferred over the devices. Task allocation aims balance the resources consumption in IoT devices while minimizing the transmission latency. These *three* factors sometime are considered together and most of the available algorithms are based on market-enable pricing schemes, which dynamically exchange the resources among the devices in IoT infrastructure by creating an artificial market [79, 210]. In ML-based IoT application, the ML models should be consider as the special tasks that are running on extremely heterogeneous computing resources in distributed manner, and these tasks usually are compute-intensive, data-intensive and network intensive. As a result, it is crucial to develop new market models to describe these special resource consumption problems and new algorithms to solve the problems.

### 5.2 Data Preprocessing

These real-world data collected from heterogeneous IoT devices usually contains outliers or is incomplete in nature, which makes it difficult to feed it into ML models directly. Data preprocessing deals with these anomalies and improves the data quality and practicality. There are several things need to be considered, namely *data cleaning*, *data annotation* and *feature engineering*. We have discussed the details of *feature engineering* in §2.3.1, will not consider in this section.

*5.2.1 Data cleaning.* Many data contains noise that is bound to confuse the ML models and reduce the accuracy of the prediction results. Data cleaning resolves this problem by completing several routine tasks such as *filling missing values*, *smoothing noise data* and *removing outliers* [4]. Empty records in the data set can be replaced manually by a specific

value, for example the attribute mean or the most common attribute in the set. It can also be marked with "unknown" or just ignored if the dataset is large enough. Noisy data, though, can be smoothed by grouping first and then averaging over each group. Data outliers can also be detected during this process if the value exceeds a predefined threshold. There are other common practices such as data normalization [203], which is used to scale all dimensions of data to a specific range. This is a very efficient method for the case where there is high variation for different dimensions of the data.

*5.2.2 Data Annotation.* As discussed in §2.1, data annotation is necessary for supervised learning based ML models, in which both the data and the corresponding target act as the input sample. The model is trained with the labeled data which is used to predict the target for a new unseen data. This is usually costly and complex due to the requirement of large volume of labeled data needed for the training. The following investigates different annotation methods that can be applied according to the size of the data to be annotated and the cost of annotation per data.

**Manual Annotation.** At the initial stage of a ML project, quick prototype of a workable model requires few labeled dataset. In this sense, the developers can manually annotate the collected data to create a small dataset. This is usually done by reviewing the data samples and attach labels following the annotation guidelines. Manual annotation by the engineers is quick and precise without any professional training, and the data quality is usually great. Problem with this approach is the lack of scalability.

**Crowdsourcing Annotation.** Crowdsourcing annotation is a scalable and cost-effective method. It is usually orchestrated by an online platform that provides access to workforce of people to complete the annotation tasks. Famous crowdsourcing platforms include Amazon Mechanical Turk (MTurk) Compared to the manual annotation, this approach can be scaled to a large dataset labelling. However, crowdsourcing method requires delicate design on quality control mechanisms to ensure the annotation quality, and the incentives or rewards for the crowds.

**Active Learning.** Active learning [232] aims to design a system capable of choosing and learning from less training data while still achieving the same or even higher accuracy. An active learning system consists two components: a *learning module* that trains a model with the current training sample and a *sample selection module* that selects the most informative samples from the unlabeled samples. Then the selected samples will be annotated manually and added to the training set. The iterative process continues until the training converges. The key is the sample selection module which can be approximately subdivided into *five* categories below according to the selection criteria [276].
*Risk Reduction* methods directly measures the performance of learning algorithm. Each time when the data is labeled, the risk is estimated according to different algorithms. The data that gives the most risk reduction is added to the training set. This approach is computationally expensive since it needs to evaluate the risk reduction each time the data is added to the training set [220].
*Uncertainty* criterion states that the most uncertain samples should be selected. That is to say, the dataset close to the decision boundary is more important for a learning algorithm. The uncertainty is typically measured by entropy, where many works have been done for better estimation [123] [286].
*Diversity* criteria considers batch training in ML algorithms. It is based on the assumption that sample diversity in each batch can accelerates the training process [123], as highly-diversified sample contain pattern that are more representative of the actual data pattern than less-diversified ones. Several researches [113] [62] are proposed for better search of the diversified data samples.

*Density* metric measures the number of unlabeled samples close to an unlabeled sample. Based on the assumption that samples in high density regions are more representative, they are less likely to be outliers and are thus more valuable in determining the decision boundary. Usually, the dataset can be grouped into clusters from which the data near the centroids is selected [188] [201].
*Relevance* criteria is similar to the *uncertainty* methods in selecting the relevant or irrelevant samples according to how they are associated with the concept. However, in many cases the relevance-based algorithms are more effective [10, 92] due to the fact that the relevant samples contribute more than the irrelevant sample for training some models. SVM, for example, the decision boundaries are only decided by a few data points.

We have discussed how these five selection criteria work individually. It is difficult, though, to contrast them in terms of performance as it depends on the specific tasks and learning algorithms. In fact, combinations of these criteria are often implemented. In [277], *uncertainty*, *diversity*, *density* and *relevance* are combined for multi-modality video annotation. Similarly, work [113] combines *uncertainty*, *diversity* and *density* metrics and the evaluation proves the combination perform well on medical image classification tasks.

### 5.3 Data Fusion

Data fusion aims to combine the data from multiple sources to provide more accurate and useful information. It offers numerous advantages for ML-based application by enhancing the data quality (finding the missing values), detecting any anomalies, conducting the prediction and finding any correlations among the distributed dataset [23, 294, 313]. However, there are multiple challenges in combining heterogeneous IoT data [3] such as data frequency, data imperfection, data correlation, data alignment and dynamic iterative process. To handle these challenges effectively, numerous data fusion methods are available in the literature. It is mainly categorized into *three* groups as given below.

*5.3.1 Probabilistic Data Fusion Algorithms.* This group consists of the algorithms that uses density function or probability distributions as a core method for data fusion. It include Bayesian techniques [24], Markov models [136], evidential reasoning [291]and other methods. These methods are simple and widely used in different applications to express the co-relation and dependency between numerous datasets. However, there are certain drawbacks with probabilistic data fusion methods highlighted in [3]. First, it can not scale with the size and modality of the data. Second, uncertain and noisy data can not be handled properly. Finally, prior probabilities and density functions are difficult to be obtained.

*5.3.2 Knowledge-based Data Fusion Algorithms.* To overcome the uncertainty of data and to increase the accuracy of fusion methods, knowledge-based data fusion methods are proposed. This method accumulates knowledge from the inexplicit data and apply over the fusion process. Different aggregation techniques and ML methods are used for the data fusion process. For example, [20, 173] (supervised learning method) and [85, 306] (unsupervised learning method) are used to discover the distribution of the complex datasets. However, the complexity of this type of methods is higher than the probabilistic methods. This class of methods, thus, may consume more computing resources and cost more time to process.

*5.3.3 Evidence-based Data Fusion Algorithms.* This group of methods are based on Demster-Shafer Theorem (DST) and recursive operations. As compared to probabilistic methods, where there are only two states (happening or non-happening) of an event, DST includes an unknown state to capture the real-world uncertainty. [122, 224] are the applications of DST for data fusion. However, increasing the data evidence also increases the complexity of this method. Therefore, this method is not suitable for the applications running on less powerful computing resources.

### 5.4 Discussion

In this section, we reviewed core components in the data acquisition process and discussed how they can attribute to generation of high quality, ready-to-use data for IoT-ML application. Additionally, there are *two* research directions can be considered to leverage others' efforts, thereby improving the performance of training a ML model. First, *Data reuse*, with the scaling of the data volume, past data is stored and usually ignored after use. However, it can be reused and mined for more values. For example they can be used for boosting semi-supervised data annotation [315], or they can be integrated with newly collected data for model training. Second, *Data re-organization*, there exists datasets for different tasks in similar areas. They may not be the same, but can be re-organized to extract the common distributions. Proper identification and extraction can be explored in this direction to save effort on data collection.

## 6 CONCLUSION

Growing numbers of internet-connected things (IoT) produce vast amounts of data, build applications and provide various services in domains such as smart cities, energy, mobility, and smart transportation. ML is becoming a preliminary technique for analyzing IoT data. It produces high-level abstraction and insight that is fed to the IoT systems for fine-tuning and improvement of the services. In this survey, we reviewed the characteristics of IoT development lifecycle and the role of ML for individual steps. In specific, we divided the development lifecycles into different modules and presented a novel taxonomy to characterize and analyze various techniques used to build an ML-based IoT application. In summary, this survey seeks to provide systematic and insightful information for researchers. It assists the development of future orchestration solutions by providing a holistic view on the current status of ML-based IoT application development, deriving key open research issues that were identified based on our critical review.

## REFERENCES


[1] A. Abdiansah and R. Wardoyo. 2015. Time complexity analysis of support vector machines (SVM) in LibSVM. *International journal computer and application* (2015).
[2] S. Achleitner, T. La Porta, et al. 2017. Adversarial network forensics in software defined networking. In *Proceedings of the Symposium on SDN Research*. ACM, 8–20.
[3] F. Alam, R. Mehmood, et al. 2017. Data fusion and IoT for smart ubiquitous environments: A survey. *IEEE Access* 5 (2017), 9533–9554.
[4] S. A. Alasadi and W. S. Bhaya. 2017. Review of data preprocessing techniques in data mining. *Journal of Engineering and Applied Sciences* 12, 16 (2017), 4102–4107.
[5] Y. Alsouda, S. Pllana, et al. 2019. IoT-based Urban Noise Identification Using Machine Learning: Performance of SVM, KNN, Bagging, and Random Forest. In *Proceedings of the International Conference on Omni-Layer Intelligent Systems*. ACM, 62–67.
[6] D. Amodei, S. Ananthanarayanan, et al. 2016. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*.
[7] F. A. Aoudia, M. Gautier, et al. 2018. RLMan: an energy manager based on reinforcement learning for energy harvesting wireless sensor networks. *IEEE Transactions on Green Communications and Networking* 2, 2 (2018), 408–417.
[8] J. Appleyard, T. Kocisky, et al. 2016. Optimizing performance of recurrent neural networks on gpus. *arXiv preprint arXiv:1604.01946* (2016).
[9] G. Ateniese, G. Felici, et al. 2013. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *arXiv* (2013).
[10] S. Ayache and G. Quénot. 2007. Evaluation of active learning strategies for video indexing. *Signal Processing: Image Communication* 22, 7-8 (2007), 692–704.
[11] B. Baker, O. Gupta, et al. 2017. Accelerating neural architecture search using performance prediction. *arXiv preprint arXiv:1705.10823* (2017).
[12] A. K. Balan, V. Rathod, et al. 2015. Bayesian dark knowledge. In *Advances in Neural Information Processing Systems*. 3438–3446.
[13] P. Baldi. 2012. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*. 37–49.
[14] R. Battiti. 1994. Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on neural networks* 5, 4 (1994), 537–550.
[15] A. Beck, A. Nedić, et al. 2014. An $o(1/k)$ gradient method for network resource allocation problems. *IEEE Transactions on Control of Network Systems* 1, 1 (2014), 64–73.
[16] G. M. Benedek and A. Itai. 1991. Learnability with respect to fixed distributions. *Theoretical Computer Science* 86, 2 (1991), 377–389.
[17] Y. Bengio. 2012. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*. Springer, 437–478.
[18] J. S. Bergstra, R. Bardenet, et al. 2011. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*. 2546–2554.
[19] J. L. Berral, Í. Goiri, et al. 2010. Towards energy-aware scheduling in data centers using machine learning. In *Proceedings of the 1st International Conference on energy-Efficient Computing and Networking*. ACM, 215–224.
[20] B. Bigdeli, F. Samadzadegan, et al. 2015. Fusion of hyperspectral and LIDAR data using decision template-based fuzzy multiple classifier system. *International Journal of Applied Earth Observation and Geoinformation* 38 (2015), 309–320.
[21] B. Biggio, B. Nelson, et al. 2012. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389* (2012).
[22] B. Biggio, K. Rieck, et al. 2014. Poisoning behavioral malware clustering. In *Proceedings of the 2014 workshop on artificial intelligent and security workshop*. ACM, 27–36.
[23] F. H. Bijarbooneh, W. Du, et al. 2016. Cloud-assisted data fusion and sensor selection for internet of things. *IEEE Internet of Things Journal* 3, 3 (2016), 257–268.
[24] T. A. Biresaw, A. Cavallaro, et al. 2015. Tracker-level fusion for robust Bayesian visual tracking. *IEEE Transactions on Circuits and Systems for Video Technology* 25, 5 (2015).
[25] A. Blumer, A. Ehrenfeucht, et al. 1987. Occam's razor. *Information processing letters* 24, 6 (1987), 377–380.

[26] J. Boedecker, J. T. Springenberg, et al. 2014. Approximate real-time optimal control based on sparse gaussian process models. In *2014 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE, 1–8.

[27] M. Boehm, M. W. Dusenberry, et al. 2016. Systemml: Declarative machine learning on spark. *Proceedings of the VLDB Endowment* 9, 13 (2016), 1425–1436.

[28] L. Bottou. 2010. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*. Springer, 177–186.

[29] O. Bousquet, S. Boucheron, et al. 2003. Introduction to statistical learning theory. In *Summer School on Machine Learning*. Springer, 169–207.

[30] D. Brauckhoff, K. Salamatian, et al. 2009. Applying PCA for traffic anomaly detection: Problems and solutions. In *IEEE INFOCOM 2009*. IEEE, 2866–2870.

[31] L. Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[32] L. Breiman. 2017. *Classification and regression trees*. Routledge.

[33] U. Breitenbücher, K. Képes, et al. 2017. âĂđDeclarative vs. Imperative: How to Model the Automated Deployment of IoT Applications?âĂIJ. *Proceedings of the 11th Advanced Summer School on Service Oriented Computing* (2017), 18–27.

[34] A. Brock, T. Lim, et al. 2017. SMASH: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344* (2017).

[35] C. BuciluǍ, R. Caruana, et al. 2006. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM.

[36] B. Cabé, E. I. W. Group, et al. 2018. IoT Developer Survey 2018. *SlideShare, April* 13 (2018).

[37] M. Castro, B. Liskov, et al. 1999. Practical Byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186.

[38] Z. B. Celik, L. Babun, et al. 2018. Sensitive information tracking in commodity IoT. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 1687–1704.

[39] K. Chaudhuri and C. Monteleoni. 2009. Privacy-preserving logistic regression. In *Advances in neural information processing systems*. 289–296.

[40] N. V. Chawla, N. Japkowicz, et al. 2004. Special issue on learning from imbalanced data sets. *ACM Sigkdd Explorations Newsletter* 6, 1 (2004), 1–6.

[41] K. Chellapilla, S. Puri, et al. 2006. High performance convolutional neural networks for document processing. In *10th International Workshop on Frontiers in Handwriting Recognition*. Suvisoft.

[42] J. Chen, X. Pan, et al. 2016. Revisiting distributed synchronous SGD. *arXiv preprint arXiv:1604.00981* (2016).

[43] L. Chen, Y. Zhang, et al. 2018. Detection of DNS DDoS Attacks with Random Forest Algorithm on Spark. *Procedia computer science* 134 (2018), 310–315.

[44] T. Chen, T. Moreau, et al. 2018. {TVM}: An Automated End-to-End Optimizing Compiler for Deep Learning. In *13th USENIX Symposium on OSDI*. 578–594.

[45] X. Chen, C. Liu, et al. 2017. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526* (2017).

[46] S. Chetlur, C. Woolley, et al. 2014. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759* (2014).

[47] T. Chilimbi, Y. Suzue, et al. 2014. Project adam: Building an efficient and scalable deep learning training system. In *11th USENIX Symposium on OSDI*. 571–582.

[48] F. Chollet. 2017. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1251–1258.

[49] P. Chopra and S. K. Yadav. 2015. Fault detection and classification by unsupervised feature extraction and dimensionality reduction. *Complex & Intelligent Systems* (2015).

[50] C.-T. Chu, S. K. Kim, et al. 2007. Map-reduce for machine learning on multicore. In *Advances in neural information processing systems*. 281–288.

[51] K. Chua, R. Calandra, et al. 2018. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*. 4754–4765.

[52] A. Coates, B. Huval, et al. 2013. Deep learning with COTS HPC systems. In *International conference on machine learning*. 1337–1345.

[53] T. Cohen and M. Welling. 2016. Group equivariant convolutional networks. In *International conference on machine learning*. 2990–2999.

[54] G. Cormode, M. Garofalakis, et al. 2011. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends® in Databases* 4, 1–3 (2011), 1–294.

[55] C. Cortes and V. Vapnik. 1995. Support-vector networks. *Machine learning* 20, 3 (1995), 273–297.

[56] M. Courbariaux, Y. Bengio, et al. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in NIPS*.

[57] M. Courbariaux, I. Hubara, et al. 2016. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv* (2016).

[58] T. M. Cover, P. Hart, et al. 1967. Nearest neighbor pattern classification. (1967).

[59] D. Crankshaw, G.-E. Sela, et al. 2018. InferLine: ML Inference Pipeline Composition Framework. *arXiv preprint arXiv:1812.01776* (2018).

[60] D. Crankshaw, X. Wang, et al. 2017. Clipper: A low-latency online prediction serving system. In *14th USENIX Symposium on NSDI*. 613–627.

[61] A. C. F. da Silva, U. Breitenbücher, et al. 2017. Internet of Things Out of the Box: Using TOSCA for Automating the Deployment of IoT Environments.. In *CLOSER*. 330–339.

[62] C. K. Dagli, S. Rajaram, et al. 2006. Leveraging active learning for relevance feedback using an information theoretic diversity measure. In *CIVR*. Springer, 123–132.

[63] M. Dash and H. Liu. 2003. Consistency-based search in feature selection. *Artificial intelligence* 151, 1-2 (2003), 155–176.

[64] J. Dean, G. Corrado, et al. 2012. Large scale distributed deep networks. In *Advances in neural information processing systems*. 1223–1231.

[65] M. Deisenroth and C. E. Rasmussen. 2011. PILCO: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*. 465–472.

[66] S. Devi and T. Neetha. 2017. Machine Learning based traffic congestion prediction in a IoT based Smart City. (2017).

[67] G. Diamos, S. Sengupta, et al. 2016. Persistent rnns: Stashing recurrent weights on-chip. In *International Conference on Machine Learning*. 2024–2033.

[68] M. Ding and H. Tian. 2016. PCA-based network traffic anomaly detection. *Tsinghua Science and Technology* 21, 5 (2016), 500–509.

[69] J.-D. Dong, A.-C. Cheng, et al. 2018. Dpp-net: Device-aware progressive search for pareto-optimal neural architectures. In *Proceedings of the ECCV*. 517–531.

[70] W. Dong, C. Chen, et al. 2013. D2: Anomaly detection and diagnosis in networked embedded systems by program profiling and symptom mining. In *RTSS*. IEEE, 202–211.

[71] W. Du and Z. Zhan. 2002. Building decision tree classifier on private data. In *Proceedings of the IEEE international conference on Privacy, security and data mining-Volume 14*. Australian Computer Society, Inc., 1–8.

[72] J. Duchi, E. Hazan, et al. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12, Jul (2011), 2121–2159.

[73] S. Egea, A. R. Mañez, et al. 2017. Intelligent IoT traffic classification using novel search strategy for fast-based-correlation feature selection in industrial environments. *IEEE Internet of Things Journal* 5, 3 (2017), 1616–1624.

[74] A. Elk. 2019. Distributed Machine Learning Toolkit: BIG DATA, BIG MODEL, FLEXIBILITY, EFFICIENCY. Retrieved March 7, 2019 from http://www.dmtk.io

[75] Y. Elovici, A. Shabtai, et al. 2007. Applying machine learning techniques for detection of malicious code in network traffic. In *Annual Conference on Artificial Intelligence*. Springer, 44–50.

[76] M. F. Elrawy, A. I. Awad, et al. 2018. Intrusion detection systems for IoT-based smart environments: a survey. *Journal of Cloud Computing* 7, 1 (2018), 21.

[77] T. Elsken, J. H. Metzen, et al. 2018. Multi-objective architecture search for cnns. *arXiv preprint arXiv:1804.09081* 2 (2018).

[78] T. Erpek, Y. E. Sagduyu, et al. 2018. Deep learning for launching and mitigating wireless jamming attacks. *TCCN* (2018).

[79] S. Eswaran, A. Misra, et al. 2012. Utility-based bandwidth adaptation in mission-oriented wireless sensor networks. *TOSN* (2012).

[80] M. Everingham, L. Van Gool, et al. 2010. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision* 88, 2 (June 2010), 303–338.

[81] M. Fredrikson, S. Jha, et al. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1322–1333.

[82] G. Frewat, C. Baroud, et al. 2016. Android voice recognition application with multi speaker feature. In *2016 18th MELECON*. IEEE, 1–5.

[83] N. Friedman, D. Geiger, et al. 1997. Bayesian network classifiers. *Machine learning* 29, 2-3 (1997), 131–163.

[84] S. Fujimoto, H. van Hoof, et al. 2018. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477* (2018).

[85] C. E. Fuss, A. A. Berg, et al. 2016. DEM Fusion using a modified k-means clustering algorithm. *International journal of digital earth* 9, 12 (2016), 1242–1255.

[86] Y. Gal, R. McAllister, et al. 2016. Improving PILCO with Bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, Vol. 4.

[87] P. Garraghan, R. Yang, et al. 2018. Emergent failures: Rethinking cloud reliability at scale. *IEEE Cloud Computing* 5, 5 (2018), 12–21.

[88] D. E. Goldberg. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning* (1st ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[89] M. Goldstein and S. Uchida. 2016. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PloS one* 11, 4 (2016), e0152173.

[90] I. Goodfellow, J. Pouget-Abadie, et al. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[91] I. J. Goodfellow, J. Shlens, et al. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).

[92] P. H. Gosselin and M. Cord. 2004. A comparison of active classification methods for content-based image retrieval. In *CVDB*. ACM, 51–58.

[93] P. Goyal, P. Dollár, et al. 2017. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677* (2017).

[94] J. Granjal, E. Monteiro, et al. 2010. A secure interconnection model for IPv6 enabled wireless sensor networks. In *2010 IFIP Wireless Days*. IEEE, 1–6.

[95] A. Graves, A. rahman Mohamed, et al. 2013. Speech recognition with deep recurrent neural networks. *ICASSP* (2013), 6645–6649.

[96] T. Gu, B. Dolan-Gavitt, et al. 2017. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733* (2017).

[97] Y. Guan and T. Plötz. 2017. Ensembles of deep lstm learners for activity recognition using wearables. *IMWUT* 1, 2 (2017), 11.

[98] S. Gupta, A. Agrawal, et al. 2015. Deep learning with limited numerical precision. In *International Conference on Machine Learning*. 1737–1746.

[99] R. Gutierrez-Osuna. 2002. Pattern analysis for machine olfaction: a review. *IEEE Sensors journal* (2002).

[100] T. Haarnoja, A. Zhou, et al. 2018. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905* (2018).

[101] H. HaddadPajouh, A. Dehghantanha, et al. 2018. A deep Recurrent Neural Network based approach for Internet of Things malware threat hunting. *Future Generation Computer Systems* 85 (2018), 88–96.

[102] M. A. Hall. 2000. Correlation-based feature selection of discrete and numeric class machine learning. (2000).

[103] Y. Han, D. Hubczenko, et al. 2019. Adversarial Reinforcement Learning under Partial Observability in Software-Defined Networking. *arXiv:1902.09062* (2019).

[104] P. Hansen, N. Mladenović, et al. 2010. Variable neighbourhood search: methods and applications. *Annals of Operations Research* 175, 1 (2010), 367–407.

[105] J. A. Hartigan and M. A. Wong. 1979. Algorithm AS 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28, 1 (1979).

[106] K. He, G. Gkioxari, et al. 2017. Mask R-CNN. *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), 2980–2988.

[107] N. Heess, S. Sriram, et al. 2017. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286* (2017).

[108] V. Hema. 2015. DoS Attack Detection Based on Naive Bayes Classifier. (2015).

[109] G. Hinton. 2019. RMSprop. Retrieved June 7, 2019 from http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[110] G. Hinton, O. Vinyals, et al. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* (2015).

[111] Q. Ho, J. Cipar, et al. 2013. More effective distributed ml via a stale synchronous parallel parameter server. In *Advances in neural information processing systems*. 1223–1231.

[112] S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[113] S. C. Hoi, R. Jin, et al. 2006. Batch mode active learning and its application to medical image classification. In *Proceedings of the 23rd ICML*. ACM, 417–424.

[114] A. G. Howard, M. Zhu, et al. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[115] C.-H. Hsu, S.-H. Chang, et al. 2018. Monas: Multi-objective neural architecture search using reinforcement learning. *arXiv preprint arXiv:1806.10332* (2018).

[116] T. Hu and Y. Fei. 2010. QELAR: A machine-learning-based adaptive routing protocol for energy-efficient and lifetime-extended underwater sensor networks. *TMC* (2010).

[117] G. Huang, S. Liu, et al. 2018. Condensenet: An efficient densenet using learned group convolutions. In *Proceedings of the IEEE Conference on CVPR*. 2752–2761.

[118] S. Ioffe and C. Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167* (2015).

[119] J. Ivaneckỳ and S. Mehlhase. 2012. An in-car speech recognition system for disabled drivers. In *International Conference on Text, Speech and Dialogue*. Springer, 505–512.

[120] G. Jagannathan, K. Pillaipakkamnatt, et al. 2009. A practical differentially private random decision tree classifier. In *2009 IEEE International Conference on Data Mining Workshops*. IEEE, 114–121.

[121] D. Jakubovitz, R. Giryes, et al. 2018. Generalization error in deep learning. *arXiv preprint arXiv:1808.01174* (2018).

[122] H. Jamshidi, T. Lukaszewicz, et al. 2011. Fusion of digital map traffic signs and camera-detected signs. In *ICSPCS*. IEEE, 1–7.

[123] A. J. Joshi, F. Porikli, et al. 2009. Multi-class active learning for image classification. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2372–2379.

[124] D. Jouan-Rimbaud, D.-L. Massart, et al. 1995. Genetic algorithms as a tool for wavelength selection in multivariate calibration. *Analytical Chemistry* 67, 23 (1995), 4295–4301.

[125] A. P. Kale and S. P. Sonavane. 2019. IoT based Smart Farming: Feature subset selection for optimized high-dimensional data using improved GA based approach for ELM. *Computers and Electronics in Agriculture* 161 (2019), 225–232.

[126] D. P. Kingma and J. Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[127] M. M. Kirmani and S. I. Ansarullah. [n. d.]. Prediction of Heart Disease using Decision Tree a Data Mining Technique. ([n. d.]).

[128] J. Ko and D. Fox. 2009. GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models. *Autonomous Robots* 27, 1 (2009), 75–90.

[129] C. Kolias, V. Kolias, et al. 2017. TermID: A distributed swarm intelligence-based approach for wireless intrusion detection. *ISECURE* (2017).

[130] V. R. Konda and J. N. Tsitsiklis. 2000. Actor-critic algorithms. In *Advances in neural information processing systems*. 1008–1014.

[131] W. Kong, Z. Y. Dong, et al. 2017. Short-term residential load forecasting based on LSTM recurrent neural network. *IEEE Transactions on Smart Grid* 10, 1 (2017), 841–851.

[132] N. Kothari, T. Millstein, et al. 2008. Deriving state machines from TinyOS programs using symbolic execution. In *IPSN*. IEEE Computer Society.

[133] A. Krizhevsky. 2014. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997* (2014).

[134] A. Krogh and J. A. Hertz. 1992. A simple weight decay can improve generalization. In *Advances in neural information processing systems*. 950–957.

[135] V. Krunic, E. Trumpler, et al. 2007. NodeMD: Diagnosing node-level faults in remote wireless sensor systems. In *MobiSys*. ACM, 43–56.

[136] P. Kumar, H. Gauba, et al. 2017. Coupled HMM-based multi-sensor data fusion for sign language recognition. *Pattern Recognition Letters* 86 (2017), 1–8.

[137] A. Kurakin, I. Goodfellow, et al. 2016. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236* (2016).

[138] S. Lakshminarasimman, S. Ruswin, et al. 2017. Detecting DDoS attacks using decision tree algorithm. In *2017 Fourth International Conference on Signal Processing, Communication and Networking (ICSCN)*. IEEE, 1–6.

[139] T. Lappas and K. Pelechrinis. 2007. Data mining techniques for (network) intrusion detection systems. *CSE at UC Riverside* (2007).

[140] Q. V. Le, M. Ranzato, et al. 2011. Building high-level features using large scale unsupervised learning. *arXiv preprint arXiv:1112.6209* (2011).

[141] V. Lebedev, Y. Ganin, et al. 2014. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553* (2014).

[142] V. Lebedev and V. Lempitsky. 2016. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[143] W. Lee, S. J. Stolfo, et al. 2000. Adaptive intrusion detection: A data mining approach. *Artificial Intelligence Review* 14, 6 (2000), 533–567.

[144] S. Levine and P. Abbeel. 2014. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*. 1071–1079.

[145] H. Li, W. Ouyang, et al. 2016. Multi-bias non-linear activation in deep neural networks. In *International conference on machine learning*. 221–229.

[146] J. Li, H. Gao, et al. 2018. Deep reinforcement learning based computation offloading and resource allocation for MEC. In *WCNC*. IEEE, 1–6.

[147] L. Li, Y. Lv, et al. 2016. Traffic signal timing via deep reinforcement learning. *IEEE/CAA Journal of Automatica Sinica* 3, 3 (2016), 247–254.

[148] M. Li, D. G. Andersen, et al. 2014. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on OSDI*. 583–598.

[149] M. Li, D. G. Andersen, et al. 2014. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*.

[150] R. Li, C. Zhang, et al. 2019. Learning Driven Mobility Control of Airborne Base Stations in Emergency Networks. *ACM SIGMETRICS Performance Evaluation Review* (2019).

[151] W. Li, P. Yi, et al. 2014. A new intrusion detection system based on KNN classification algorithm in wireless sensor network. *Journal of Electrical and Computer Engineering* 2014 (2014).

[152] T. P. Lillicrap, J. J. Hunt, et al. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[153] J. Lin, W. Yu, et al. 2017. A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet of Things Journal* 4, 5 (2017), 1125–1142.

[154] R. Lioutikov, A. Paraschos, et al. 2014. Sample-based informationl-theoretic stochastic optimal control. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 3896–3902.

[155] C. Liu, B. Zoph, et al. 2018. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 19–34.

[156] H. Liu, K. Simonyan, et al. 2017. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436* (2017).

[157] L. Liu, M. Zhang, et al. 2014. A survey on workflow management and scheduling in cloud computing. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 837–846.

[158] M. Lopez-Martin, B. Carro, et al. 2017. Conditional variational autoencoder for prediction and feature recovery applied to intrusion detection in iot. *Sensors* (2017).

[159] G. Louppe. 2014. Understanding random forests: From theory to practice. *arXiv preprint arXiv:1407.7502* (2014).

[160] Q. Lu, T. Peng, et al. 2010. Utility-based resource allocation in uplink of OFDMA-based cognitive radio networks. *International Journal of Communication Systems* 23, 2 (2010), 252–274.

[161] M.-T. Luong, H. Pham, et al. 2015. Effective approaches to attention-based neural machine translation. *EMNLP* (2015).

[162] C. Lyu, K. Huang, et al. 2015. A unified gradient regularization family for adversarial examples. In *2015 IEEE International Conference on Data Mining*. IEEE, 301–309.

[163] N. Ma, X. Zhang, et al. 2018. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

[164] D. J. MacKay and D. J. Mac Kay. 2003. *Information theory, inference and learning algorithms*. Cambridge university press.

[165] R. Madan, J. Borran, et al. 2010. Cell association and interference coordination in heterogeneous LTE-A cellular networks. *IEEE Journal on selected areas in communications* 28, 9 (2010), 1479–1489.

[166] R. Madan, S. P. Boyd, et al. 2010. Fast algorithms for resource allocation in wireless cellular networks. *IEEE/ACM Transactions on Networking (TON)* 18, 3 (2010), 973–984.

[167] H. Mao, M. Alizadeh, et al. 2016. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM.

[168] H. Mao, R. Netravali, et al. 2017. Neural adaptive video streaming with pensieve. In *SIGCOMM*. ACM.

[169] A. Marcano-Cedeno, J. Quintanilla-Domínguez, et al. 2010. Feature selection using sequential forward selection and classification applying artificial metaplasticity neural network. In *IECON*. IEEE.

[170] M. Mathieu, M. Henaff, et al. 2013. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851* (2013).

[171] Y. Meidan, M. Bohadana, et al. 2017. Detection of unauthorized iot devices using machine learning techniques. *arXiv preprint arXiv:1709.04647* (2017).

[172] X. Meng, J. Bradley, et al. 2016. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research* 17, 1 (2016), 1235–1241.

[173] A. Merentitis and C. Debes. 2015. Automatic fusion and classification using random forests and features extracted with deep learning. In *IGARSS*. IEEE, 2943–2946.

[174] A. Meunkaewjinda, P. Kumsawat, et al. 2008. Grape leaf disease detection from color imagery using hybrid intelligent system. In *ECTI-CON*, Vol. 1. IEEE, 513–516.

[175] A. Mirhoseini, H. Pham, et al. 2017. Device placement optimization with reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2430–2439.

[176] T. M. Mitchell. 1999. Machine learning and data mining. *Commun. ACM* 42, 11 (1999), 30–36.

[177] V. Mnih, A. P. Badia, et al. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.

[178] V. Mnih, K. Kavukcuoglu, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[179] M. Mohammadi, A. Al-Fuqaha, et al. 2018. Deep learning for IoT big data and streaming analytics: A survey. *IEEE Communications Surveys & Tutorials* 20, 4 (2018).

[180] P. Moritz, R. Nishihara, et al. 2018. Ray: A distributed framework for emerging {AI} applications. In *13th USENIX Symposium on OSDI*. 561–577.

[181] S. Mukherjee and N. Sharma. 2012. Intrusion detection using naive Bayes classifier with feature reduction. *Procedia Technology* 4 (2012), 119–128.

[182] M. S. Munir, S. F. Abedin, et al. 2017. RNN based Energy Demand Prediction for Smart-Home in Smart-Grid Framework.

[183] R. Munos, T. Stepleton, et al. 2016. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*. 1054–1062.

[184] A. Murad, F. A. Kraemer, et al. 2019. Autonomous Management of Energy-Harvesting IoT Nodes Using Deep Reinforcement Learning. *arXiv:1905.04181* (2019).

[185] A. Nagabandi, G. Kahn, et al. 2018. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 7559–7566.

[186] P. Naraei, A. Abhari, et al. 2016. Application of multilayer perceptron neural networks and support vector machines in classification of healthcare data. In *FTC*. 848–852.

[187] B. Neyshabur, S. Bhojanapalli, et al. 2017. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*. 5947–5956.

[188] H. T. Nguyen and A. Smeulders. 2004. Active learning using pre-clustering. In *Proceedings of the twenty-first international conference on Machine learning*. ACM, 79.

[189] J. Nocedal and S. Wright. 2006. *Numerical optimization*. Springer Science & Business Media.

[190] C. Noel and S. Osindero. 2014. Dogwild!-distributed hogwild for cpu & gpu. In *NIPS Workshop on Distributed Machine Learning and Matrix Computations*.

[191] D. Oh and I. Yun. 2018. Residual error based anomaly detection using auto-encoder in smd machine sound. *Sensors* 18, 5 (2018), 1308.

[192] X. Pan, M. Lam, et al. 2016. Cyclades: Conflict-free asynchronous machine learning. In *Advances in Neural Information Processing Systems*. 2568–2576.

[193] N. Papernot, P. McDaniel, et al. 2017. Practical black-box attacks against machine learning. In *ASIACCS 2017*. ACM, 506–519.

[194] N. Papernot, P. McDaniel, et al. 2016. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE.

[195] G. I. Parisi, R. Kemker, et al. 2019. Continual lifelong learning with neural networks: A review. *Neural Networks* (2019).

[196] R. Parr and S. J. Russell. 1998. Reinforcement learning with hierarchies of machines. In *NIPS*.

[197] S. Peddabachigari, A. Abraham, et al. [n. d.]. Intrusion detection systems using decision trees and support vector machines. ([n. d.]).

[198] K. Peng, V. Leung, et al. 2018. Intrusion detection system based on decision tree over big data in fog environment. *Wireless Communications and Mobile Computing* 2018 (2018).

[199] Z. Peng, X. Chen, et al. 2018. AXNet: ApproXimate computing using an end-to-end trainable neural network. In *ICCAD*. ACM, 11.

[200] H. Pham, M. Y. Guan, et al. 2018. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268* (2018).

[201] G.-J. Qi, Y. Song, et al. 2006. Video annotation by active learning and cluster tuning. In *2006 Conference on CVPRW'06*. IEEE.

[202] N. Qian. 1999. On the momentum term in gradient descent learning algorithms. *Neural networks* 12, 1 (1999), 145–151.

[203] J. Quackenbush. 2002. Microarray data normalization and transformation. *Nature genetics* 32, 4s (2002), 496.

[204] J. R. Quinlan. 1979. Discovering rules by induction from large collections of examples. *Expert systems in the micro electronics age* (1979).

[205] J. R. Quinlan. 1986. Induction of decision trees. *Machine learning* 1, 1 (1986), 81–106.

[206] J. R. Quinlan. 2014. *C4. 5: programs for machine learning*. Elsevier.

[207] R. Raina, A. Madhavan, et al. 2009. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual ICML*. ACM.

[208] N. Ramakrishnan and T. Soni. 2018. Network Traffic Prediction Using Recurrent Neural Networks. In *2018 17th IEEE ICMLA*. IEEE.

[209] B. Randell. 1975. System structure for software fault tolerance. *Ieee transactions on software engineering* 2 (1975), 220–232.

[210] S. Rangwala, R. Gummadi, et al. 2006. Interference-aware fair rate control in wireless sensor networks. In *ACM SIGCOMM Computer Communication Review*. ACM.

[211] T. Rao, N. Rajasekhar, et al. [n. d.]. An efficient approach for weather forecasting using support vector machines.

[212] M. Rastegari, V. Ordonez, et al. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*. Springer.

[213] E. Real, A. Aggarwal, et al. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

[214] B. Recht, C. Re, et al. 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*. 693–701.

[215] K. Rieck, P. Trinius, et al. 2011. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security* 19, 4 (2011), 639–668.

[216] H. Robbins and S. Monro. 1951. A stochastic approximation method. *The annals of mathematical statistics* (1951), 400–407.

[217] H. Röger and R. Mayer. 2019. A Comprehensive Survey on Parallelization and Elasticity in Stream Processing. *ACM Computing Surveys (CSUR)* 52, 2 (2019), 36.

[218] R. Roman, J. Zhou, et al. 2013. On the features and challenges of security and privacy in distributed internet of things. *Computer Networks* 57, 10 (2013), 2266–2279.

[219] A. Romero, N. Ballas, et al. 2014. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550* (2014).

[220] N. Roy and A. McCallum. 2001. Toward optimal active learning through monte carlo estimation of error reduction. *ICML, Williamstown* (2001), 441–448.

[221] B. I. Rubinstein, P. L. Bartlett, et al. 2009. Learning in a large function space: Privacy-preserving mechanisms for SVM learning. *arXiv preprint arXiv:0911.5708* (2009).

[222] B. I. Rubinstein, B. Nelson, et al. 2009. Antidote: understanding and defending against poisoning of anomaly detectors. In *SIGCOMM on Internet measurement*. ACM.

[223] F. Ruelens, B. J. Claessens, et al. 2016. Residential demand response of thermostatically controlled loads using batch reinforcement learning. *IEEE Transactions on Smart Grid* 8, 5 (2016), 2149–2159.

[224] V. Saeidi, B. Pradhan, et al. 2014. Fusion of airborne lidar with multispectral spot 5 image for enhancement of feature extraction using dempster–shafer theory. *TGRS* (2014).

[225] P. Samangouei, M. Kabkab, et al. 2018. Defense-gan: Protecting classifiers against adversarial attacks using generative models. *arXiv preprint arXiv:1805.06605* (2018).

[226] M. Sandler, A. Howard, et al. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[227] B. Sanjaa and E. Chuluun. 2013. Malware detection using linear SVM. In *Ifost*, Vol. 2. IEEE, 136–138.

[228] S. Santurkar, D. Tsipras, et al. 2018. How does batch normalization help optimization?. In *Advances in Neural Information Processing Systems*. 2483–2493.

[229] T. Schaul, J. Quan, et al. 2015. Prioritized experience replay. *arXiv preprint arXiv:1511.05952* (2015).

[230] J. Schulman, S. Levine, et al. 2015. Trust region policy optimization. In *International Conference on Machine Learning*. 1889–1897.

[231] J. Schulman, F. Wolski, et al. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[232] B. Settles. 2012. Active learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 6, 1 (2012), 1–114.

[233] A. Shafahi, W. R. Huang, et al. 2018. Poison frogs! targeted clean-label poisoning attacks on neural networks. In *Advances in Neural Information Processing Systems*.

[234] U. Shaham, Y. Yamada, et al. 2018. Understanding adversarial training: Increasing local stability of supervised models through robust optimization. *Neurocomputing* (2018).

[235] W. Shang, K. Sohn, et al. 2016. Understanding and improving convolutional neural networks via concatenated rectified linear units. In *ICML*. 2217–2225.

[236] T. A. Shinde and J. R. Prasad. 2017. IoT based animal health monitoring with naive Bayes classification. *IJETT* 1, 2 (2017).

[237] J. Shlens. 2014. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100* (2014).

[238] R. Shokri and V. Shmatikov. 2015. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. ACM, 1310–1321.

[239] R. Shokri, M. Stronati, et al. 2017. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 3–18.

[240] A. Shustanov and P. Yakimov. 2017. CNN design for real-time traffic sign recognition. *Procedia engineering* 201 (2017), 718–725.

[241] S. Sicari, A. Rizzardi, et al. 2015. Security, privacy and trust in Internet of Things: The road ahead. *Computer networks* 76 (2015), 146–164.

[242] L. Sifre and S. Mallat. 2014. Rigid-motion scattering for image classification. *Ph. D. dissertation* (2014).

[243] A. K. Sikder, H. Aksu, et al. 2017. 6thsense: A context-aware sensor-based attack detector for smart devices. In *26th {USENIX} Security Symposium*. 397–414.

[244] K. J. Singh and D. S. Kapoor. 2017. Create Your Own Internet of Things: A survey of IoT platforms. *IEEE Consumer Electronics Magazine* 6, 2 (2017), 57–68.

[245] S. Singh and I. Chana. 2016. A survey on resource scheduling in cloud computing: Issues and challenges. *Journal of grid computing* 14, 2 (2016), 217–264.

[246] S. P. Singh. 1992. Reinforcement learning with a hierarchy of abstract models. In *Proceedings of the National Conference on Artificial Intelligence*. JOHN WILEY & SONS LTD.

[247] S. L. Smith and Q. V. Le. 2017. A bayesian perspective on generalization and stochastic gradient descent. *arXiv preprint arXiv:1710.06451* (2017).

[248] E. R. Sparks, A. Talwalkar, et al. 2015. Automating model search for large scale machine learning. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*. ACM, 368–380.

[249] N. Srivastava, G. Hinton, et al. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* 15, 1 (2014), 1929–1958.

[250] J. Steinhardt, P. W. W. Koh, et al. 2017. Certified defenses for data poisoning attacks. In *Advances in neural information processing systems*. 3517–3529.

[251] I. Stoica, D. Song, et al. 2017. A berkeley view of systems challenges for ai. *arXiv preprint arXiv:1712.05855* (2017).

[252] J. Su and H. Zhang. 2006. A fast decision tree learning algorithm.

[253] M.-Y. Su. 2011. Real-time anomaly detection systems for Denial-of-Service attacks by weighted k-nearest-neighbor classifiers. *Expert Systems with Applications* 38, 4 (2011), 3492–3498.

[254] O. Suciu, R. Marginean, et al. 2018. When does machine learning {FAIL}? generalized transferability for evasion and poisoning attacks. In *USENIX Security 18)*. 1299–1316.

[255] S. Sultana, D. Midi, et al. 2014. Kinesis: a security incident response and prevention system for wireless sensor networks. In *SenSys*. ACM.

[256] S. Sun, W. Chen, et al. 2017. Ensemble-compression: A new method for parallel training of deep neural networks. In *ECML PKDD*. Springer.

[257] R. B. S. Swetha and K. G. Meena. [n. d.]. Smart Grid-A Network based Intrusion Detection System. *International Journal of Computer Applications* 975 ([n. d.]), 8887.

[258] C. Szegedy, S. Ioffe, et al. 2017. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[259] M. Tan, B. Chen, et al. 2019. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*.

[260] R. Tandon, Q. Lei, et al. 2017. Gradient coding: Avoiding stragglers in distributed learning. In *International Conference on Machine Learning*. 3368–3376.

[261] S. Thrun and A. Schwartz. 1995. Finding structure in reinforcement learning. In *NIPS*.

[262] J. T. Townsend. 1971. Theoretical analysis of an alphabetic confusion matrix. *Perception & Psychophysics* 9, 1 (1971), 40–50.

[263] F. Tramèr, A. Kurakin, et al. 2017. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204* (2017).

[264] F. Tramèr, F. Zhang, et al. 2016. Stealing machine learning models via prediction apis. In *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 601–618.

[265] A. Ullah, J. Ahmad, et al. 2018. Action Recognition in Video Sequences using Deep Bi-Directional LSTM With CNN Features. *IEEE Access* 6 (2018), 1155–1166.

[266] M. Vacher, B. Lecouteux, et al. 2015. Speech and speaker recognition for home automation: Preliminary results. In *SpeD*. IEEE.

[267] L. G. Valiant. 1984. A theory of the learnable. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*. ACM, 436–445.

[268] H. Van Hasselt, A. Guez, et al. 2016. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[269] V. Vanhoucke, A. Senior, et al. 2011. Improving the speed of neural networks on CPUs. (2011).

[270] V. Vapnik. 2013. *The nature of statistical learning theory*. Springer science & business media.

[271] S. Venugopalan, H. Xu, et al. 2014. Translating videos to natural language using deep recurrent neural networks. *arXiv preprint arXiv:1412.4729* (2014).

[272] S. A. Vinterbo. 2012. Differentially private projected histograms: Construction and use for prediction. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 19–34.

[273] R. Volpi, H. Namkoong, et al. 2018. Generalizing to unseen domains via adversarial data augmentation. In *Advances in Neural Information Processing Systems*. 5334–5344.

[274] G. Wang, J. Hao, et al. 2010. A new approach to intrusion detection using Artificial Neural Networks and fuzzy clustering. *Expert systems with applications* (2010).

[275] J. Wang and L. Perez. 2017. The effectiveness of data augmentation in image classification using deep learning. *Convolutional Neural Networks Vis. Recognit* (2017).

[276] M. Wang and X.-S. Hua. 2011. Active learning in multimedia annotation and retrieval: A survey. *ACM Transactions on Intelligent Systems and Technology (TIST)* (2011).

[277] M. Wang, X. S. Hua, et al. 2007. Interactive video annotation by multi concept multi modality active learning. *ICSC* (2007).

[278] Z. Wang, T. Schaul, et al. 2015. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581* (2015).

[279] C. J. Watkins and P. Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.

[280] D. Weerasiri, M. C. Barukh, et al. 2017. A taxonomy and survey of cloud resource orchestration techniques. *ACM Computing Surveys (CSUR)* 50, 2 (2017), 26.

[281] W. Wen, C. Wu, et al. 2016. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*. 2074–2082.

[282] Z. Wen, P. Bhatotia, et al. 2018. Approxiot: Approximate analytics for edge computing. In *ICDCS*. IEEE.

[283] Z. Wen, D. OâĂŹNeill, et al. 2015. Optimal demand response using device-based reinforcement learning. *IEEE Transactions on Smart Grid* 6, 5 (2015), 2312–2324.

[284] Z. Wen, R. Yang, et al. 2017. Fog orchestration for internet of things services. *IEEE Internet Computing* 21, 2 (2017), 16–24.

[285] B. Wu, F. Iandola, et al. 2017. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 129–137.

[286] Y. Wu, I. Kozintsev, et al. 2006. Sampling strategies for active learning in personal photo retrieval. In *2006 IEEE International Conference on Multimedia and Expo*. IEEE.

[287] C. Xiao, B. Li, et al. 2018. Generating adversarial examples with adversarial networks. *arXiv preprint arXiv:1801.02610* (2018).

[288] H. Xiao, H. Xiao, et al. 2012. Adversarial Label Flips Attack on Support Vector Machines.. In *ECAI*. 870–875.

[289] E. P. Xing, Q. Ho, et al. 2015. Petuum: A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data* 1, 2 (2015), 49–67.

[290] W. Xu, D. Evans, et al. 2017. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155* (2017).

[291] X. Xu, J. Zheng, et al. 2017. Data classification using evidence reasoning rule. *Knowledge-Based Systems* 116 (2017), 144–151.

[292] Z. Xu, Y. Wang, et al. 2017. A deep reinforcement learning based framework for power-efficient resource allocation in cloud RANs. In *ICC*. IEEE.

[293] N. J. Yadwadkar, B. Hariharan, et al. 2016. Multi-task learning for straggler avoiding predictive job scheduling. *The Journal of Machine Learning Research* 17, 1 (2016), 3692–3728.

[294] C. Yang, L. Feng, et al. 2018. A novel data fusion algorithm to combat false data injection attacks in networked radar systems. *TSIPN* (2018).

[295] J. Yang, Y. Chen, et al. 2008. Detecting sybil attacks inwireless and sensor networks using cluster analysis. In *2008 5th IEEE International Conference on Mobile Ad Hoc and Sensor Systems*. IEEE, 834–839.

[296] M. Ye and E. Abbe. 2018. Communication-computation efficient gradient coding. *arXiv preprint arXiv:1802.03475* (2018).

[297] E. Yigitoglu, M. Mohamed, et al. 2017. Foggy: a framework for continuous automated IoT application deployment in fog computing. In *AIMS*. IEEE, 38–45.

[298] S. Zagoruyko and N. Komodakis. 2016. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. *arXiv preprint arXiv:1612.03928* (2016).

[299] S. Zhai, Y. Cheng, et al. 2016. Doubly convolutional neural networks. In *Advances in neural information processing systems*. 1082–1090.

[300] Z.-H. Zhan, X.-F. Liu, et al. 2015. Cloud computing resource scheduling and a survey of its evolutionary approaches. *ACM Computing Surveys (CSUR)* 47, 4 (2015), 63.

[301] C. Zhang, A. Kumar, et al. 2016. Materialization optimizations for feature selection workloads. *ACM Transactions on Database Systems (TODS)* 41, 1 (2016), 2.

[302] C. Zhang, P. Patras, et al. 2019. Deep learning in mobile and wireless networking: A survey. *IEEE Communications Surveys & Tutorials* (2019).

[303] J. Zhang, Z. Zhang, et al. 2012. Functional mechanism: regression analysis under differential privacy. *Proceedings of the VLDB Endowment* 5, 11 (2012), 1364–1375.

[304] W. Zhang, W. Guo, et al. 2018. LSTM-based analysis of industrial IoT equipment. *IEEE Access* 6 (2018), 23551–23560.

[305] W. Zhang, S. Gupta, et al. 2015. Staleness-aware async-sgd for distributed deep learning. *arXiv preprint arXiv:1511.05950* (2015).

[306] W.-A. Zhang, B. Chen, et al. 2016. Hierarchical fusion estimation for clustered asynchronous sensor networks. *IEEE Trans. Automat. Control* 61, 10 (2016), 3064–3069.
[307] X. Zhang, X. Zhou, et al. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*. 6848–6856.
[308] C. Zhao, C. Chen, et al. 2018. Application of Auxiliary Classifier Wasserstein Generative Adversarial Networks in Wireless Signal Classification of Illegal Unmanned Aerial Vehicles. *Applied Sciences* 8, 12 (2018), 2664.
[309] C. Zhao, M. Shi, et al. 2018. Research on the open-categorical classification of the Internet-of-things based on generative adversarial networks. *Applied Sciences* (2018).
[310] D. Zhao, Y. Chen, et al. 2016. Deep reinforcement learning with visual attention for vehicle classification. *IEEE Transactions on Cognitive and Developmental Systems* (2016).
[311] W. Zhao, S. Tang, et al. [n. d.]. An improved kNN algorithm based on essential vector. *Elektronika ir Elektrotechnika* 123, 7 ([n. d.]), 119–122.
[312] X. Zhao, Z. Gao, et al. 2011. A fault detection algorithm based on cluster analysis in wireless sensor networks. In *2011 Seventh International Conference on Mobile Ad-hoc and Sensor Networks*. IEEE, 354–355.
[313] Y. Zheng, X. Yi, et al. 2015. Forecasting fine-grained air quality based on big data. In *SIGKDD (KDD)*. ACM.
[314] B. Zhou, J. Cao, et al. 2010. Adaptive traffic light control in wireless sensor network-based intelligent transportation system. In *VTC*. IEEE, 1–5.
[315] S. Zhou, Q. Chen, et al. 2013. Active deep learning method for semi-supervised sentiment classification. *Neurocomputing* 120 (2013), 536–546.
[316] Y. Zhou, S. Ebrahimi, et al. 2018. Resource-efficient neural architect. *arXiv preprint arXiv:1806.07912* (2018).

# A APPENDIX A

| Software Specification | **Cloud** | **Edge** | **IoT devices** |
|---|---|---|---|
| Main OS | - Ubuntu<br>- CentOS<br>- Debian<br>- RHEL<br>- Windows Server<br>- Amazon Linux | - Raspbian<br>- NOOBS<br>- Amazon FreeRTOS<br>- RIOT<br>- Google Fuchsia OS<br>- Windows 10 IoT | - Amazon FreeRTOS<br>- Contiki<br>- TinyOS<br>- RIOT<br>- Ubuntu Core<br>- Mbed OS |
| Programming Language | - Java<br>- ASP.NET<br>- Python<br>- PHP<br>- Ruby | - Java<br>- Python<br>- C<br>- C++<br>- JavaScript | - C<br>- C++<br>- Java<br>- JavaScript<br>- Python |
| Platforms | - AWS<br>- Azure<br>- Google Cloud Platform<br>- IBM Cloud<br>- Oracle Cloud | - Amazon Greengrass<br>- EdgeX<br>- Cisco IOx<br>- Akraino Edge Stack<br>- Eclipse ioFog | - AWS IoT<br>- Azure IoT<br>- GCP IoT<br>- IBM Watson<br>- Cisco IoT cloud connect |

Table 3. List of OS, programming language and platform in IoT layers

# B APPENDIX B

## B.1 Traditional Machine Learning (TML) methods

In this subsection, we give the details of several TML algorithms, as well as their IoT applications.

***Logistic Regression (LR).*** Logistic regression is a linear classifier capable of performing binary or multi-class classification. It is among the simplest classification algorithms. In a binary classification setting, prediction target $y$ is usually formulated as $y \in \{0, 1\}$ and the prediction probability of the positive class given a d-dimensional input $\mathbf{x} = [x_1, x_2, ..., x_d] \in \mathbb{R}^d$ can be calculated via:

$$p(y = 1|\mathbf{x}; \boldsymbol{\theta}) = \sigma(\mathbf{w}^T\mathbf{x} + b) = \sigma(\sum_{i=1}^{d} w_i x_i + b), \tag{1}$$

In the equation $\boldsymbol{\theta} = \{\mathbf{w} = [w_1, w_2, ..., w_d] \in \mathbb{R}^d, b \in \mathbb{R}\}$ denotes the model parameters, and $\sigma(\cdot)$ is an activation function used to squash the linear output within the range $[0, 1]$ for probabilistic interpretation. The training of the classifier aims

to learn suitable values for the parameters $\boldsymbol{\theta} = \{\mathbf{w}, b\}$, starting from some random initialization, through minimizing of a loss or cost function $J(\boldsymbol{\theta})$. For LR, log (Cross-Entropy) loss is used, i.e., $J(\boldsymbol{\theta}) = -\ln p(y|\mathbf{x};\boldsymbol{\theta})$ for data point $\{\mathbf{x}, y\}$ to facilitate the calculation of the model gradient and as to minimize the model loss. With the trained model, for any query data the classification decision can be made via thresholding the predicted probability.

Extend the binary LR to support $c$-class prediction scenarios, the target $y \in \{1, 2, ..., c\}$. Softmax function is applied to the output layer to normalize the $c$ outputs into probabilities. Different from binary LR, there are $c$ sets of parameters $\boldsymbol{\theta} = \{\mathbf{W} = [\mathbf{w}_1, ..., \mathbf{w}_c]^T, \mathbf{b} = [b_1, ..., b_c]^T\}$ to be estimated by minimizing the cross-entropy loss (i.e., log loss). With the trained model parameters, at the inference stage the class label will be assigned to the one with the largest classification probability.

***Decision Trees (DT)***. Decision Tree is a tree-like model for classification or regression tasks. A decision tree is constructed by nodes and edges where node can be seen as a feature, and the edge represents a condition for classification. The learning process of DT is to select the optimal feature that can best split the training examples in a recursive manner. Based on certain criteria such as information gain [205] or Gini impurity [71], the root node can be selected from the features, which will divide the whole training population into two or more homogeneous sets. For each sub-population, sub-nodes will be selected in a similar manner and this process will repeat until all the subsets are pure (i.e., with the same class label for each subset). These nodes and edges constitute the trained model and can be used for inference.

Different from most of the other classifiers (such as LR, SVM, ANN, DL) which require the input to be normalised to numerical values, the unique tree-structure of DT make it possible to take both numerical and nominal values, making it a highly interpretable tool for various classification tasks (e.g., medical records diagnosis). However, DT suffers from the "curse of dimensionality", and it faces overfitting problem when the input dimensionality is too high. When with unstructured high-dimensional data, feature engineering/extraction is one of the necessary steps to take (for dimensionality reduction), before DT is applied.

DT can be used as a main classifier (e.g., for low-dimensional structured data) or collaborative classifier with other machine learning algorithms on various IoT applications, such as Intrusion detection system [198] in fog environment.

***Random Forest (RF)***. As its name implies, a RF is an ensemble model with many DTs as base classifiers. The individual DTs are constructed by random sampling the features and the training examples for diversity, boosting the performance of multiple classifier systems. The random sampling process makes the individual DTs less correlated — with different prediction errors— and the aggregating function can smooth the large prediction variance, making RF a robust classifier with high generalisation capabilities. There are several key hyper-parameters for RF, and two main ones are: feature number for individual DTs, and number of DT classifiers. For individual DTs, there is a trade-off between generalization and discrimination capabilities with respect to feature dimension, and one popular heuristic value is to use the square root of original dimension number (e.g., the default setting in scikit-learn). For the RF though, the performance of the model usually correlates wit the number of DTs. Yet the performance gain tends to become less significant since the diversity (among the DT classifiers, which are correlated to some extent) will decrease accordingly. Since both the efficiency and storage are proportional (in a linear manner) to the classifier number, to find a number configuration that can result in optimal effectiveness, storage and efficiency performance is of vital importance.

RF is one of the most popular classifiers due to its great generalization capability. For example, it has been used for intrusion detection [43, 171] and anomaly detection. In a previous study [171], the authors collected and manually labeled data from 17 distinct IoT devices and use RF algorithm to recognize IoT device categories from the white list.

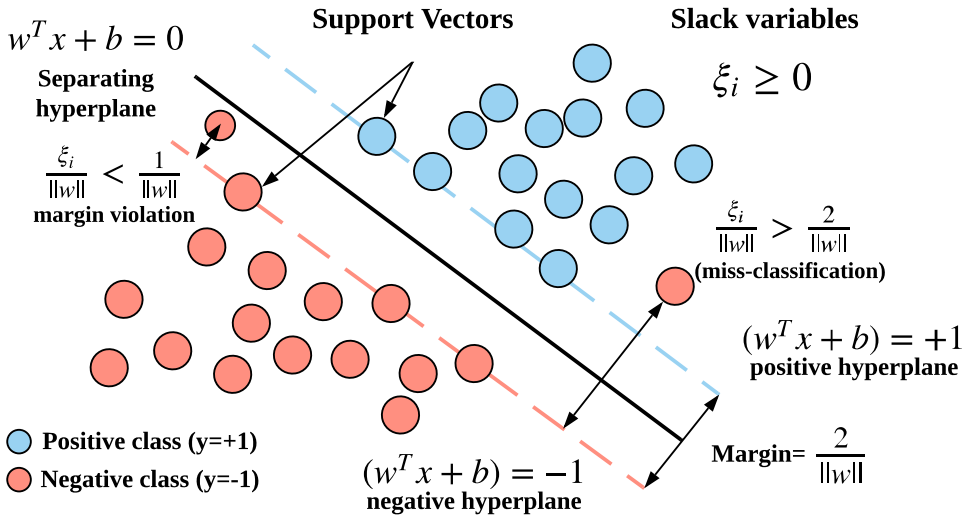


Fig. 11. Basic concepts in SVM

***Support Vector Machine (SVM).*** Support Vector Machine (SVM), also referred to as large margin classifier, aims to find a decision boundary (separating hyperplane) that can best separate (i.e., with the largest margin) positive/negative classes. Fig. 11 shows some basic concepts in SVM including support vectors, margin and slack variable $\xi$. $\xi$ is a non-negative variable that is used to measure the misclassified instances and those within the margin (i.e., margin violation, as shown in Fig. 11). The objective function of (soft-margin) SVM can be constructed to maximize the margin while penalizing these instances:

$$\underset{\mathbf{w},b,\xi_i}{\text{argmin}} \frac{||\mathbf{w}||^2}{2} + C\sum_i^N \xi_i, \quad \text{subject to} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \text{where} \quad \xi_i \geq 0, \quad i = 1, 2, ..., N. \tag{2}$$

In Eq (2), $\{\mathbf{x}_i, y_i\}_{i=1}^N$ are the $N$ training sets with $y_i \in \{-1, 1\}$, and $C$ is a regularization hyper-parameter that trade-off between the margin and errors (i.e., measured by $\sum_i^N \xi_i$). It is obvious to see that if we set $C = 0$, we can get a classifier with the large margin at a cost of potential high training errors. On the other hand, if we set $C$ a very large number (or $\infty$ in theory), errors are less likely to be tolerant, and we may end up with a classifier with narrow or hard margin. It is worth noting that Eq.(2) can be further simplified into $\text{argmin}_{\mathbf{w},b} \frac{||\mathbf{w}||^2}{2} + C\sum_i^N \max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))$, where the term $\max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b))$ is also referred to as hinge loss.

On the other hand, for highly non-linearly separable data, instead of employing feature engineering/extraction process, an elegant alternative—kernel SVM can be applied, and Radial Basis Function (RBF) is one of the most popular kernel functions. Kernel SVM also has the aforementioned characteristics such as soft margin, and it tends to have great performance on small non-linearly separable data. Fig. **??** shows the margins on linearly separable(by linear SVM) and non-linearly separable(by kernel SVM) data, respectively.

SVM is notable for its generalization capability and suitable for those small datasets with high-dimensional features [270], and there are many IoT applications such as Android malware detection system [227], smart weather prediction [211], etc.

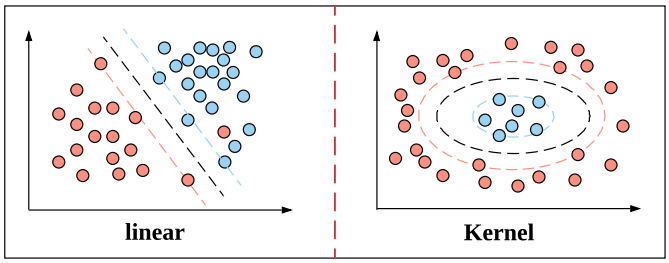


Fig. 12. Margins for linear (left) and kernel (right) SVM

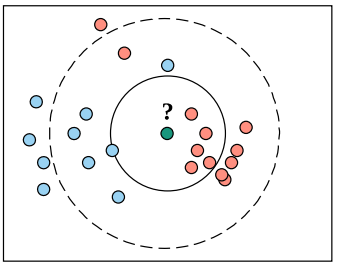


Fig. 13. Majority voting process in KNN

***K*-Nearest Neighbour (KNN).** KNN is a nonparametric, instance-based, non-linear classifier. Under the classification setting, given any query data, KNN essentially performs majority voting among the $K$ most similar training samples, as shown in Fig.**??**. The similarity can be measured by some distance metrics such as Minkowski distance $D(\mathbf{x}, \mathbf{y}) = (\sum_{j=1}^{d} |x_j - y_j|^p)^{\frac{1}{p}}$ (for $d$-dimensional vectors $\mathbf{x}, \mathbf{y}$). It is worth noting that when $p = 1$ and $p = 2$, the Minkowski distance can be seen as Manhattan Distance and Euclidean distance respectively, yet for different applications the optimal distance metric may vary from case to case.

One of the key property is that KNN does not require any training process (i.e., lazy learning), and for any query data, the distance calculation has to be performed for each sample in the whole training set, which makes KNN a less-scalable approach for large datasets. Another issue is the selection of hyper-parameter $K$. A small $K$ may make KNN sensitive to outliers in the training set while a large one may make KNN less discriminate. Nevertheless, KNN is a powerful non-linear classifier when with low-dimensional small datasets, and there are many IoT applications such as network intrusion detection [151], anomaly detection [253] and Urban noise identification [5], etc.

***Naive Bayes (NB).*** NB is a probabilistic classifier which takes the class prior distribution into account, and assumes the features are conditionally independent. Based on the bayesian theory and the posterior probability, it can be presented as $p(y|\mathbf{x}) = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})}$, where $p(\mathbf{x}|y)$ is the likelihood; $p(y)$ is the prior; $p(\mathbf{x})$ is the evidence. The classification process is to assign the label with the largest posterior probability, and in this case the term evidence $p(\mathbf{x})$ remains a constant which can be cancelled out, i.e., $p(y|\mathbf{x}) \propto p(\mathbf{x}|y)p(y)$, which can be further written into $p(y|\mathbf{x}) \propto \prod_{j=1}^{d} p(x_j|y)p(y)$ for d-dimensional input $\mathbf{x}$ due to the feature independence assumption. In practice, logarithm is often used to avoid the problem of floating point underflow, and the class label $\hat{y}$ can be assigned via:

$$\hat{y} = \underset{y \in \{1,2,\ldots,c\}}{\operatorname{argmax}} \ln p(y) + \sum_{j=1}^{d} \ln p(x_j|y), \tag{3}$$

NB is good at modelling both continuous and discrete data. For example the likelihood of a discrete feature can be calculated by frequency while the likelihood of a continuous feature can be calculated by density estimation (e.g., Gaussian). It takes the prior of class distribution into consideration, which is helpful in data imbalanced problems. However, it also suffers from the 'curse of dimensionality' like DT, and normally can't be used directly on unstructured data before feature extraction/engineering approaches have been applied. When the feature independence assumption is not significantly violated, it is normally served as an efficient and effective classifier. There are many IoT applications, such as network traffic analysis for DoS attach detection [108], animal health monitoring [236], etc.

Table 4. Summary of Traditional Machine Learning Models(Note: In time complexity, $m$ represents the number of training sample, $n$ represents the feature dimension, $k$ represents the selected $K$ value, $c$ represents the class number and $t$ indicates the tree number)

| Method | Learning model | Category | Typical input data | Time Complexity | Characteristics | IoT Application |
|---|---|---|---|---|---|---|
| DT | Supervised | Discriminative | Various | $O(m \cdot n^2)$ [252] | • Dividing training samples to branches and leaves<br>• High interpretability method<br>• Require large memory space due to the construction nature. | • Intrusion detection [197]<br>• Suspicious detection on traffic sources [75]<br>• Future Heart Attack Quantity prediction [127] |
| SVM | | Discriminative | Various | $O(m^2 \cdot n)$[50] | • Good Generalization capability and suitability for small dataset with large feature<br>• Difficult on selection optimal kernel<br>• High computation complexity on large dataset with complex kernel | • Malware detection [227]<br>• Attacks detection in smart grids [257]<br>• Smart weather prediction [211] |
| NB | | Discriminative | Various | $O(m \cdot n + n \cdot c)$ [50] | • Ease to implement<br>• Generalization well to multi-class problem<br>• Low dependence on large dataset and robustness<br>• Hard to capture relation information | • Detection of network intrusion [181]<br>• Animal health monitoring [236] |
| KNN | | Discriminative | Various | $O(m \cdot n \cdot k)$ [311] | • Nonparametric, instance-based<br>• No training process, no model construction<br>• Sensitive to the outlier<br>• Difficult on selection optimal K | • Anomalies detection [89]<br>• Urban noise identification [5] |
| RF | | Discriminative | Various | $O(t \cdot n^2 \cdot logn)$ [159] | • Robust to over-fitting<br>• Bypasses feature selection<br>• Impractical in specific real-time application | • DDoS attacks detection [138]<br>• Unauthorized IoT devices detection [171] |
| K-Means | Unsupervised | Clustering | Various | $O(m \cdot n \cdot k)$[50] | • Ease to use on Unlabelled data<br>• Produce tighter cluster than hierarchical clustering<br>• Less effective than supervised learning method<br>• Difficult on selection optimal K | • Sensor fault detection [312]<br>• Sybil detection in industrial WSNs [295] |
| PCA | | Dimension Reduction | Various | $O(m \cdot n^2 + n^3)$ [50] | • Used for dimensional reduction<br>• Consequently reduce the complexity of the model<br>• Should be used with other ML methods | • Real-time detection systems in IoT environments [76]<br>• Traffic anomaly detection [68] |

***K-Means***. Different from the supervised classification models above, K-means is an unsupervised clustering algorithm without using class label information for training. Given a number of data points, K-Means aims to find $K$ centroids (i.e., means), and the corresponding nearest samples to form the clusters. Various distance metrics can be used for K-means algorithm, and the most common one is based on Euclidean distance, whose objective function is:

$$\text{argmin} \sum_{i=1}^{N} \sum_{k=1}^{K} \gamma_{ik} ||\mathbf{x}_i - \mu_k||^2, \tag{4}$$

where $\mu_k$ is the centroid of the $k$th cluster and $\gamma_{ik} \in \{0, 1\}$ denotes whether sample $\mathbf{x}_i$ belongs to the $k$th cluster (1) or not (0). K-means clustering is an heuristic process—starting from random values (of the centroids $\{\mu_k\}_{k=1}^{K}$), it will 1) assign each example to the nearest cluster and 2) update the $K$ centroids (by re-calculating the means of the corresponding samples for each cluster). It is an iterative process and the updating will stop until Eq.(4) is minimized (e.g., lower than a pre-defined threshold).

K-means is among the most popular clustering algorithms due to its simplicity. However, Euclidean distance based method makes it limited to spherical datasets, and finding the most suitable distance metrics (for different applications/datasets) is one of the key issues on improving the performance of K-means algorithm. K-means is widely used for IoT applications such as sensor fault detection [312], Sybil detection [295], etc.

***Principal component analysis(PCA)***. PCA is another unsupervised learning approach and it is normally used for dimensionality reduction or feature decorrelation. Covariance matrix $\mathbf{S}$ which reflects the correlation of the features can be calculated via $\mathbf{S} = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^T$ for $N$ $d$-dimensional training data points $\{\mathbf{x}_i\}_{i=1}^{N}$ with $\mu = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i$. Eigenvalue decomposition can be performed on $\mathbf{S}$ such that the leading $s(< d)$ eigenvectors can be used as transformation

matrix for feature decorrelation or dimensionality reduction. PCA is one of the most popular feature extraction tools due to its simplicity, and there are many IoT applications such as traffic anomaly detection [30].

Table 4 summarized the aforementioned TML, including their advantages, disadvantages and applications in IoT system.

### B.2 Deep Learning (DL) methods

In this section, we detail several deep learning algorithms: Deep Neural Networks (DNN), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), etc.

***Deep Neural Networks (DNN).*** As aforementioned, LR is one of the most popular linear classifiers, which means the decision boundary is a line or hyperplane in the feature space. However, most of the data are more complex that are not linearly separable in real-world scenarios. In this case, directly applying linear classifiers such as LR on the raw data would yield unsatisfied classification performance. One could empirically use hand-crafted features followed by LR, yet this trial-and-error process can be time-consuming, and it may be challenging to define the high-order discriminant features where the data can be linearly separable. One alternative is to use data-driven methods, and it is straight-forward to extend LR to DNN (i.e., Multilayer Perceptron MLP).

DNN is built upon LR with a stack of hidden layers. By applying activation functions like sigmoid, tanh, or ReLU on the hidden layers, raw features can be transformed in a non-linear manner, yielding discriminant (i.e., linearly separable) features before the output layer. Similar to LR, the training process is to minimise the log loss. However, the parameters of DNN are the layer-wise connections, which can be learned using the back-propagation approach. DNN performs end-to-end learning, which means it learns the feature extractors and classifiers simultaneously.

DNN can be deemed as the simplest form of deep learning models which comprises one input layer, one output layer as well as multiple hidden layers for more complex feature extraction. It is worth noting that Artificial Neural Network(ANN) is a special case of DNN with only one hidden layer. The layers are organised in a hierarchical manner, with each layer being a function of the layer that preceded it. For example, the second layer $\mathbf{h}^{(2)}$ (i.e., also the first hidden layer) can be expressed as a function of the first (i.e., input) layer (e.g., with input vector $\mathbf{x} \in \mathbb{R}^d$):

$$\mathbf{h}^{(2)} = g^{(2)}(\mathbf{W}^{(2)T}\mathbf{x} + \mathbf{b}^{(2)}), \tag{5}$$

where $\{\mathbf{W}^{(2)}, \mathbf{b}^{(2)}\}$ are the parameters and $g^{(2)}$ is the activation function (e.g., ReLU) for the second layer. We also call these layers fully connected or dense layer. Similarly, the $l^{th}(l > 2)$ hidden layer can be written as

$$\mathbf{h}^{(l)} = g^{(l)}(\mathbf{W}^{(l)T}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}). \tag{6}$$

Assuming there are a total number of $L$ layers, then output layer linearly transforms the previous hidden units $\mathbf{h}^{(L-1)}$, followed by a softmax function for probability scaling:

$$p(\mathbf{y}|\mathbf{h}^{(L-1)}) = \text{softmax}(\mathbf{W}^{(L)T}\mathbf{h}^{(L-1)} + \mathbf{b}^{(L)}), \tag{7}$$

Note the model parameters include the layer-wise weight matrices and bias vectors, i.e., $\boldsymbol{\theta} = \{\mathbf{W}^{(l)}, \mathbf{b}^{(l)}\}_{l=2}^{L}$, which needed to be estimated by minimising the loss function (e.g., log loss). DNN has been widely used for various IoT applications, including wearable-based activity recognition [266], traffic congestion prediction[66], healthcare[186], etc.

***Convolutional Neural Networks (CNNs).*** Similar to the DNN, CNN also comprises multiple hidden layers. It learns to map from 2D data (e.g., images) at the input, output the class probabilities at the output. DNN usually contains three

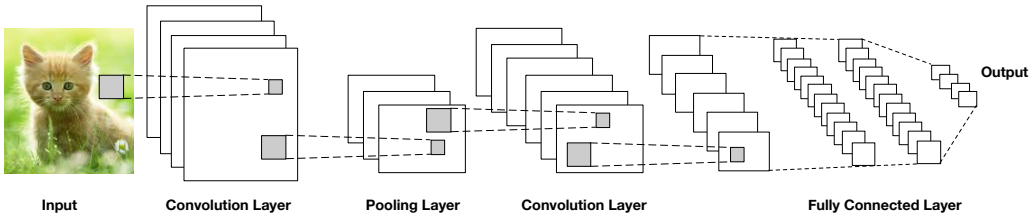


Fig. 14. An architecture of CNN

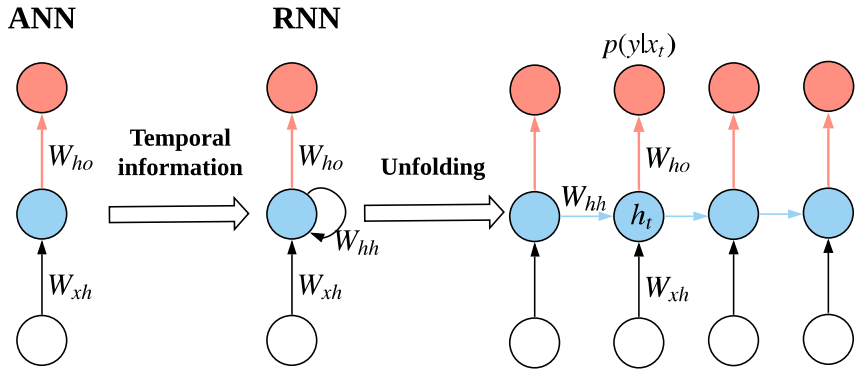


Fig. 15. From ANN to RNN

types of hidden layers: convolution layer and pooling layer at each stage and FC/dense layer right before the output layer. In convolution layer, local patterns can be extracted by learning several convolutional kernels/filters (with a predefined size, such as $3 \times 3$), each of which has the same shared weights to be estimated (via back-propagation). This weight-sharing scheme makes CNN effective in dealing with high-dimensional data (such as high-resolution images). After convolution operation with a number of kernels/filters, the corresponding feature maps can be formed, and then pooling layer can be employed for down-sampling these feature maps for more compact representation. In CNN, normally there are multiple convolution/pooling layers, before FC/dense layers can be applied, and Fig. 14 showed an example architecture of CNN. Since cameras are one of the major part of IoT environment, CNN can be a very useful tool and it has been widely used such as traffic sign detection on autonomous driving [240].

***Recurrent Neural Networks (RNNs).*** RNN is a type of deep model and it is designed to model and recognise sequential data (e.g., time-series data). Compared with DNN, the hidden units of RNN can feed-forward on itself in the next timestamp and thus can memorise the temporal information for sequential inference. Fig. 15 shows how to extend an ANN (DNN with one hidden layer) to a three-layer vanilla RNN by modelling the temporal information, and in this Figure we use arrow to represent FC/dense layer for better visual effects. Compared with ANN, RNN has an additional set of parameters, i.e., the hidden-to-hidden transformation matrix $\mathbf{W}_{hh}$ to be estimated, and the full parameters can be expressed as: $\Theta = \{\mathbf{W}_{xh}, \mathbf{W}_{hh}, \mathbf{W}_{ho}, \mathbf{b}_o, \mathbf{b}_h\}$. For $\mathbf{x}_t$, the input vector at the $t^{th}$ timestamp, the feed-forward pass can

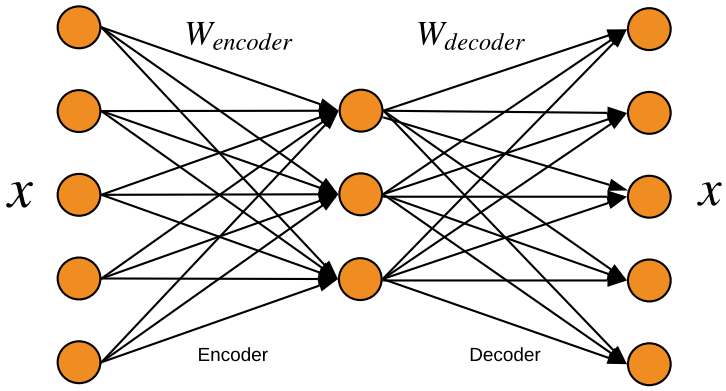


Fig. 16. The encoder-decoder structure of a three-layer Auto-encoder

be written as:

$$p(\mathbf{y}|\mathbf{x}_t) = \text{softmax}(\mathbf{W}_{ho}^T \mathbf{h}_t + \mathbf{b}_o)$$
$$\text{where} \qquad \mathbf{h}_t = \tanh(\begin{bmatrix}\mathbf{W}_{xh}\\ \mathbf{W}_{hh}\end{bmatrix}^T [\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_h), \tag{8}$$

It is clear that the current hidden state $\mathbf{h}_t$ is calculated based on the previous hidden state $\mathbf{h}_{t-1}$ and the current signal $\mathbf{x}_t$, while the output layer remains the same as DNN (see Eq.(7)). We can see RNN is an end-to-end method that both the sample-wise features $\mathbf{h}_t$ and predictions $p(\mathbf{y}|\mathbf{x}_t)$ can be learned simultaneously.

For complex time-series data analysis, we need a deeper RNN (i.e., larger window, multiple layers, etc.) to capture the high-level temporal/contextual information. Yet, vanilla RNN has a gradient explosion/vanishing problem owing to its numerical properties. In the 90s, a complex hidden unit named long short term memory (LSTM) [112] was proposed for large-scale recurrent neural network construction, which can preserve the error/gradient that can be back-propagated effectively through time and layers. LSTM includes four different gates organised in a special internal structure, in contrast to tanh function for activation in vanilla RNN. Accordingly, it also has four sets of specific gate parameters to be estimated.

RNNs achieve great performance in time-series applications, for example machine translation and speech recognition. In IoT applications, it has shown its great performance in sensor-based power station condition prediction [304], wearable-based activity recognition [**?** ], etc.

***Auto-encoder (AE).*** An Auto-encoder(AE) is a type of unsupervised neural network, and it can transform the data into latent code/representation/feature (e.g., in lower-dimensionality), from which the original data can be reconstructed. AE is widely used for data compression and feature extraction, and a typical AE includes two parts: encoder, and decoder. Fig. 16 shows a three-layer encoder-decoder structure for AE, and the $d$-dimensional input vector $\mathbf{x}$ can be transformed into the latent representation $\mathbf{h}$ via a linear transformation followed by an activation function $g(.)$, i.e., $\mathbf{h} = g(\mathbf{W}_{\text{encoder}}^T \mathbf{x} + \mathbf{b}_0)$. Similarly, decoder can be used to reconstruct the data such that $\mathbf{x}' = g(\mathbf{W}_{\text{decoder}}^T \mathbf{h} + \mathbf{b}_1)$. Note

$\Theta = \{\mathbf{W}_{\text{encoder}}, \mathbf{b}_0, \mathbf{W}_{\text{decoder}}, \mathbf{b}_1\}$ are the model parameters that can be estimated via minimising a reconstruction loss

$$\underset{\Theta}{\operatorname{argmin}} L(\mathbf{x} - \mathbf{x}'). \tag{9}$$

Note in Eq. (9), $L(.)$ can be very flexible, and some popular ones include mean squared error, log loss (for binary input), etc. It is worth noting that when with small dataset, a popular constraint is to set $\mathbf{W}_{\text{encoder}} = \mathbf{W}^T_{\text{decoder}}$, which can limit the degree of freedom of this model with better generalisation capabilities. AE can be used as a powerful tool for dimensionality reduction. Compared with PCA, the non-linear transformation may extract more discriminant information for classification. It can also be used for anomaly detection, and the intuition is that the trained AE (based on normal class) cannot reconstruct the abnormal class well, yielding large reconstruction errors. There are many IoT applications for AE, such as fault diagnosis in hardware devices[49], and anomaly detection in the performance of assembly lines[191].

***Generative Adversarial Network (GAN).*** Generative Adversarial Networks (GAN) is a generative model with adversarial architecture. Figure 17 shows the basic network architecture of a GAN, which contains a generator and a discriminator.

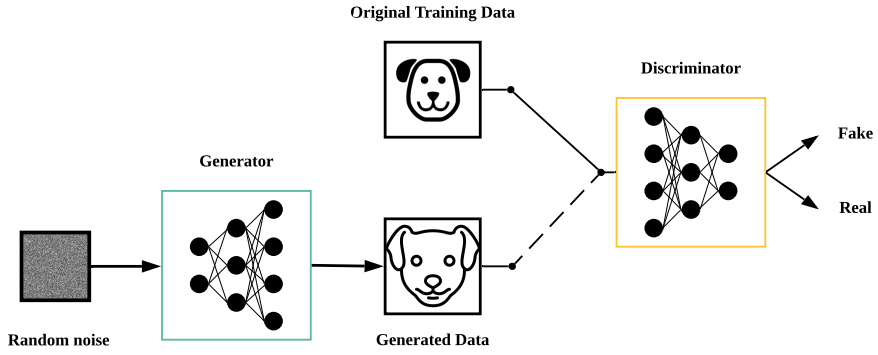


Fig. 17. Generative Adversarial Network Architecture

The generator aims at generating indistinguishable samples compared to the real data. While the discriminator works adversarially to distinguish the generated fake samples from the real data. It is an iterative competition process that will eventually lead to a state where the generated samples are indistinguishable from the real data. The generative model aims to learn the data distribution and generate data samples for those domains that lack data samples.

The decent idea of applying adversarial structure on deep neural network brought the impressive result related to content generated, and realistic visual content. Moreover, it can be applied to many applications such as image restoration, style transfer, sample generation. However, there are three main drawbacks on GAN frameworks. The first one is that the network is hard to train and the training process is destabilized (i.e., “Non-convergence"). The second drawback is that the generator will tend to produce limited varieties of samples at the end of training period (i.e., “Mode collapse"). The last drawback of GAN is the gradient diminishing problem that occurs when the balance between generator and discriminator were broken.

Table 5. Summary of Deep Machine Learning Models

| Method | Learning Type | Category | Input data Type | Characteristics | IoT Application |
|---|---|---|---|---|---|
| CNNs | Supervised | Discriminative | 2-D (image, sound, etc.) | • Mainly used on image processing<br>• Less connection compared to DNNs.<br>• Require large training samples. | • Traffic sign detection<br>• Plant disease detection<br>• Bridge Crack Detection |
| RNNs | | Discriminative | Sequential data | • Mainly used to analyse sequential data<br>• Useful in IoT applications with time-dependent data | • Identify movement pattern<br>• Behavior detection<br>• Human activity recognition<br>• Mobility prediction |
| AEs | Unsupervised | Generative | Various | • Major used for feature extraction, and dimensionality reduction<br>• Optimized by reconstructs input data<br>• Can be used on unlabeled data | • Emotion recognition<br>• Machinery fault diagnosis<br>• Intrusion detection<br>• Failure detection |
| GANs | | Generative | Various | • Learn data discribution<br>• Can be used as a data generation tool<br>• Two parts networks: a generator and a discriminator | • Localization and way finding<br>• Image to text |

GANs have been recently implemented in the IoT environment, mainly on IoT security. For example, a previous study[309] proposed a GAN based framework for improving open-categorical classification on individual identity authentication application. Moreover, GAN has been used as a tool to generate large datasets that do not need manual annotation. For example, a study[308] explored the feasibility of using GAN to generate illegal Unmanned Aerial Vehicles (UAVs) dataset and obtain a better classification model with better accuracy. As the data generated from sensors may be unlabelled, GANs may have more potential applications on the IoT environment.

### B.3 Reinforcement Learning (RL) methods

The goal of a reinforcement learning agent is to find an optimal policy to maximize the expected sum of future rewards $J(\theta)$ parameterized by $\theta$. At each time step $t$, reward $r_t = r(a_t, s_t)$ is given when an agent takes an action $a_t \in A$ at state $s_t \in S$.

$$\arg\max_{\theta} \quad J(\theta) = \mathbb{E}_{t\sim p_\theta(t)}[\sum_{t} r(s_t, a_t)] \tag{10}$$

In the equation, $p_\theta(t)$ represents the interactions between the RL agent and outer environments. The trajectory depends on two factors: the agent policy and the environment dynamic. Agent takes actions based on its policy $\pi_{\theta(a_t|s_t)}$. The dynamic transition for the environment $s_t \times a_t \rightarrow s_{t+1}$ can be expressed as $p(s_{t+1}|s_t, a_t)$ and is usually unknown. Overall, the whole trajectory $p_\theta(t)$ can be represented as:

$$P_\theta(s_1, a_1 ... s_T, a_T) = p(s_1) \prod_{t=1}^{T} \pi_{\theta(a_t|s_t)} p(s_{t+1}|s_t, a_t) \tag{11}$$

It is clear from the above equation that, if we are smart on the agent policy $\pi_{\theta(a_t|s_t)}$ or the transition dynamic $p(s_{t+1}|s_t, a_t)$, we can find the best trajectory $p_\theta(t)$ which maximises reward at each time step. The RL community has formalised these two approaches as *Model-Free* and *Model-Based* approaches based on the fact that the later one directly models the system dynamic transitions.

Fig 18 gives a broad view of the reinforcement learning world. The reinforcement learning method can be divided into *Model-free* methods and *Model-based* methods with correlations combining the features of both methods. The model-free approach can be subdivided into *Policy based*, *Value based* and *Actor-Critic* approach according to the ways

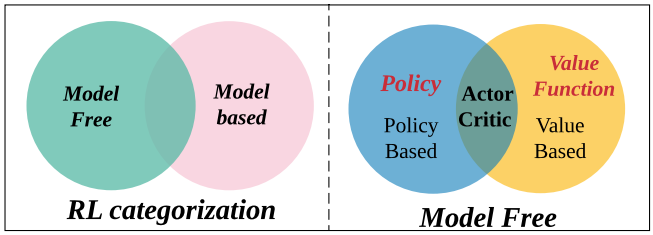

Fig. 18. Reinforcement Learning Categorization

that the best policy is generated. In this section, we will detail both approaches with simple mathematical expressions and list the most famous algorithms. We will also cover some works that take advantage of both methods.

**Model-Free Methods.** They learn a policy and decide the best action to take given a certain state. It can be categorized by *policy-gradient*, *value-based* and *actor-critic* methods, based on how the policy is generated.

*Value-Based* methods learn a value function to estimate the "goodness" $V(s_t)$ for reaching a certain state $s_t$, or the "goodness" $Q(s_t, a_t)$ for taking certain action $a_t$ given the state $s_t$. Hereby the "goodness" function estimates the sum of future rewards from current state $s_t$ till the end $s_T$ (Given a finite trajectory). At each step, the agent chooses the action with highest score based on the estimated value function. Value-based approach is deterministic, while may not be sufficient to solve complex problems. List of the most prevalent algorithms include Q-Learning [279], DQN[178], Prioritised Experience Replay[229], Dueling DQN[278], Double DQN[268] and Retrace[183].

*Policy-Gradient* methods provide an attractive paradigm by directly maximising $J(\theta)$ (Equa 10) with respect to the parameters $\theta$ of the policy $\pi_\theta(a|s)$. The gradient with respect to the parameters $\theta$ can be derived as

$$\nabla_\theta J = \mathbb{E}_\theta[\sum_t \nabla_\theta log \pi_\theta(a_t|s_t)(R_t - b_t)] \tag{12}$$

$R_t = \sum_{t'=t} \gamma^{t'-t} r(s_{t'}, a_{t'})$ with a discounting factor $\gamma$ emphasizing the agent more on recent rewards. To reduce the policy variance given different datasets, a baseline $b_t$ that does not depend on future states or actions is subtracted. In practice, the expected future return is sampled and aggregated within a trajectory. Several works have been proposed based on this paradigm to either reduce the policy variance [230], increase scalability [107] and reduce sample complexity [231].

*Actor-Critic* algorithms [130] are similar to the policy-gradient approach in updating the policy, while an estimated value function $V(s_t)$ is leveraged in replace of the constant baseline $b$ in original equation 12. The term $(R_t - b_t)$ is thus an estimate of the *Advantage* defined as $A(a_t, s_t) = Q(a_t, s_t) - V(s_t)$ where $a_t$ is the action and $s_t$ denotes the current state, with $R_t$ estimating $Q(a_t, s_t)$. Actor-critic methods experience much lower variance with the policy $\pi$ and value function $V$ seen as actors and critics respectively. Works utilising this architecture benefit from both the policy-gradient and value-based methods, with DDPG [152] combining deep Q-learning for continuous action space, A3C [177] for concurrent training. Other prevalent works include TD3 [84], Soft-Critic SAC [100].

**Model-based Methods.** The model-based algorithms differ from the model-free methods in that the later one cares less about the environment inner working and the rewards are estimated through sampling. On the contrary, model-based approach focuses on the model to predict the next state at each time step. Model-based RL achieve good sample efficiency by learning the transition dynamics of the environments directly. During learning, sample trajectories are collected and trained with supervised learning. Several different approaches have been applied to study the dynamics, Gaussian Process approaches [65][128][26], Time-varying linear models approaches [144][154] or Deep networks [185][86][51].

## C APPENDIX C

| Metrics | Formula | Type | Evaluation Focus |
|---|---|---|---|
| Accuracy(acc) | $\frac{tp+tn}{tp+fp+tn+fn}$ | Binary | This metric measures the correct percentage of the total samples |
| | $\frac{\sum_{i=1}^{l} \frac{tp_i+tn_i}{tp_i+tn_i+fp_i+tn_i}}{l}$ | Multi class | Average accuracy for all classes |
| Error Rate(err) | $\frac{fp+fn}{tp+fp+tn+fn}$ | Binary | This metric measures the miss-classification percentage over evaluated samples |
| | $\frac{\sum_{i=1}^{l} \frac{fp_i+fn_i}{tp_i+tn_i+fp_i+tn_i}}{l}$ | Multi class | Average Error Rate of all classes |
| Precision(p) | $\frac{tp}{tp+fp}$ | Binary | Precision measures correct classified positive samples in a positive class |
| | $\frac{\sum_{i=1}^{l} \frac{tp_i}{tp_i+fp_i}}{l}$ | Multi class | Average of precision on each class |
| Recall(r) | $\frac{tp}{tp+fn}$ | Binary | Recall measures the fraction of correct classified positive samples |
| | $\frac{\sum_{i=1}^{l} \frac{tp_i}{tp_i+tn_i}}{l}$ | Multi class | The average of recall for each class |
| F1-Score(FS) | $\frac{2*p*r}{p+r}$ | Binary | This metric measures the harmonic mean of recall and precision |
| | $\frac{2*p_M*r_M}{p_M+r_M}$ | Multi class | The average F1-Score |
| Geometric Mean(GM) | $\sqrt{tp*tn}$ | Binary | This metric is similar to the F1-Score but aims to maximize the $tp$ rate and $tn$ rate |

Table 6. Evaluation Metrics for Classification Problem.
Note: $i$ indicates the class $C_i$, and $M$ donates the macro-averaging.

| Metric | Formula | Eavluation Focus |
|---|---|---|
| Mean Squared Error | $\frac{1}{N}\sum_{i=1}^{N}(y_i-\hat{y})^2$ | MSE calculate the square residual for every data point, contribute more to the outlier |
| Mean Absolute Error | $\frac{1}{N}\sum_{i=1}^{N}\lvert y_i-\hat{y}\rvert$ | MAE calculate the absolute residual for every data point, so that negative and positive residuals do not cancel out |
| Mean Absolute Percentage Error | $\frac{1}{N}\sum_{i=1}^{N}\lvert\frac{y_i-\hat{y}}{y}\rvert$ | Percentage equivalent of MAE |
| Mean Percentage Error | $\frac{1}{N}\sum_{i=1}^{N}(\frac{y_i-\hat{y}}{y})$ | MPE can help to check whether the model are underestimates or overestimates |

Table 7. Evaluation Metrics for Classification Problem. Note: $y_i$ indicates the predicted value for $i$th sample, and $\hat{y}$ donates the ground truth value. $N$ indicate the total number of samples.

## D APPENDIX D

In this section, we review the defending methods for the *three* types of the attacks discussed in the manuscript.

**Defending model exploratory attack.** The most straightforward defense of this type of attack is to constrain the API, not returning confidences and not responding to probing queries. The minimization can be achieved in three approaches: *Rounding confidence*, *Differential privacy* and *Ensemble methods*.

*Rounding confidence* is a type of defenses that round confidence sources of an application to some fixed precisions [81]. Notably, some online ML service providers are already working on it. For instance, BigML and Amazon provides 5 decimal places and 16 significant digits for their confidence scores respectively when answering queries. Limiting the precision can decreases the success rate of attacks is because the outputs of the model are approximated. Equation-solving attacks, for example, if the outputs of an equation-system is rounded, it will increase the difficulty for attacker to guess the target function.

*Differential privacy* is a class of mechanisms to protect, especial, the privacy of training data [272]. A set of *Differential privacy* methods have been applied to protect regressions [39, 303], SVMs [221], decision trees [120] and neural networks [238]. The main idea of *Differential privacy* is to avoid a query that allows an adversary to distinguish closely neighboring model parameters.

*Ensemble methods* returns an aggregated outputs predicted by a set of models. This prevent method was mentioned in [264] which may have more resilience against the model exploratory attacks, compared with other methods. Bonawitz et.al, designed a new protocol to compute the sum of a large subset of models supporting a secure federated learning setting.

**Defending data poisoning attack.** Compared with other systems, defending data poisoning attacks is more critical in machine learning systems, because the training data comes from the outside world is very easy to be poisoned. Steinhardt et.al [250] developed an outlier detector for linear classifiers to generate approximate upper bounds of the training dataset. The data (poisoned) that exceeds the bounds will be removed from the training dataset, not changing the distribution of the clean data. The paper [222] aimed to defense the attacks that attempts to poison the PCA based anomaly detection models. Since the PCA methods are less sensitive to outliers, an ideal approach against the variance injection casued by the perturb data is to filter the poisoned data with predefined threshold. The paper also proposed two approaches [222] for selecting the threshold: the first uses covariance matrix, and the second is to find the maximal scale estimate of the data projection.

**Defending evasion attack.** Crafting the adversarial samples is a complex optimization process as it is very hard to build a general tool for defending. Thus, *adversarial training* which adds the adversarial examples to the training set is a straightforward way to increase the model robustness [162, 234, 263]. Moreover, Defense-GAN [225] leverages a generative model to generate more samples similar to the training data, reducing the adversarial perturbation significantly. Apart from these general defending methods, some defenses use model hardening techniques. [290] proposed a feature squeezing method to reduce the complexity of representing the data. Less important adversarial perturbations are filtered out afterwards. In the following section, we survey the attacks and the defenses in real IoT applications and then discuss the challenges of building a secure IoT application.