

Machine Learning Systems on Edge Devices, A Study of Challenges and Solutions

FIU Capstone Team 2022

Abstract

The computing world has seen a tremendous explosion of innovation thanks to the development of novel Artificial Intelligence (AI) and Machine Learning (ML) techniques. From AI agents that can win games against the most advance human players, to language models that can generate impressive content at superhuman levels. While this is exciting, most of the current efforts on these fields are focused on developing better algorithms and architectures from the theoretical side, while not much effort is put on understanding the engineering challenges of bringing AI systems to real-life, especially on edge devices. As the community of AI/ML practitioners finds exciting new applications of these techniques, we are faced with the challenge of building software systems whose behavior is dictated by data and code. Pioneers on these efforts argue that operationalizing ML systems requires its own engineering discipline (MLOps), and that techniques previously applied to build traditional software systems do not cover the complexity of this new type of software system built on data.

1 Introduction

The computing world has seen a tremendous explosion of innovation thanks to the development of novel Artificial Intelligence (AI) and Machine Learning (ML) techniques. From AI agents that can win games against the most advance human players, to language models that can generate impressive content at superhuman levels. While this is exciting, most of the current efforts on these fields are focused on developing better algorithms and architectures from the theoretical side, while not much effort is put on understanding the engineering challenges of bringing AI systems to real-life, especially on edge devices. As the community of AI/ML practitioners finds exciting new applications of these techniques, we are faced with the challenge of building software systems whose behavior is dictated by data and code. Pioneers on these efforts argue that operationalizing ML systems requires its own engineering discipline (MLOps), and that techniques previously applied to build traditional software systems do not cover the complexity of this new type of software system built on data.

The Scalable AI group consists of AI and ML practitioners as well as ML operations (MLOps) experts who work together to develop and use modern software engineering practices to deploy AI/ML solutions at scale.

Contributions

The following are the most important contributions of this work:

- Development
- Security
- Deployment

2 Development

In recent years, the interest in edge intelligence has grown exponentially worldwide. AI/Edge technology in ML (Machine Learning) research has expanded drastically and has changed our everyday lives with its performance in various fields such as facial recognition, traffic prediction, and many more. The promising solution of ML models on edge devices is to process and train data locally, instead of solely relying on cloud computing. Running the edge model on the cloud may reduce latency but proposes the sentiment of worriment as individual's privacy may be at risk of exposure. The development of ML on edge devices is not dissimilar from that of development on standard platforms, such as computers. Necessary knowledge relating to model development, model training, data collection, algorithm selection, model compression, etc, all

THIS PRELIMINARY DRAFT IS NOT APPROVED FOR PUBLIC RELEASE. ©2021 The MITRE Corporation. ALL RIGHTS RESERVED.

carry over to developing ML for edge devices. However, it is important to keep in mind what the target device is when ruminating between these various aspects of development, as while the knowledge carries over, you must keep in mind the restrictions of the device you choose to develop for. These restrictions include the memory of the device, the processing power of the device, and the device's latency. All three of these factors are important topics to consider when developing for edge devices, as they have a large impact on what models you can use and requirements for data.

2.1 Training Architecture

Training architecture vastly depends on the competence and capacity of an edge device or server. For example, an edge device that is powerful enough, could easily adopt the same training as a centralized server or solo training (training on a single device). However, if the edge device doesn't pertain to a similar capability, it would require collaborative training where the generated data is dispersed amid several devices and servers work collectively to execute training tasks. Training machine learning models, especially deep learning models, typically requires a lot of computational power and a large number of training examples[11]. Considering that an abundance of power is essential for edge devices, larger devices are typically obligatory that contain a solid computational capacity. However, the goal is to achieve a smaller device, that shares the same capabilities as the larger device. The model size needs to be kept small by using as few trainable parameters as possible, and the inference time and energy need to be kept low by minimizing the number of computations[11]. Hence, there are two kinds of training architectures: solo training, i.e., perform training tasks on a single edge device/server, and collaborative training, i.e., few devices and servers work collaboratively to perform training tasks[17]. Subsequently, solo training has a greater constraint on the hardware, which is typically available less often, most of the research is done on collaborative training because it entails less requirements, meaning more devices can be of use.

2.2 Solo Training

This training architecture is considered ideal as it reinforces itself with the data, generates it, and trains itself, and doesn't need assistance from other. However, centralized edge devices can execute the data stored and train based off it without the need for collaborative training. Edge devices that perform solo training would be harder to maintain as not every device will be capable of doing solo training unless the device is very powerful. Since the storage capacity of edge devices is limited, the content replacement must be considered[17]. Therefore, the best choice is collaborative training when considering the two architectures.

2.3 Collaborative Training

This training architecture is the one that is the most reachable with the current technological advances, it is quicker, and a wide variety of devices would be capable of holding the collaborative training since the hardware is not required to be as powerful. The most common collaborative training architecture is Federated Learning. Federated learning shows illustrates where a server employs multiple devices and allocates training tasks for them[17]. Another example of a widely utilized collaborative training is called Edge2Train, which is another way to conduct edge training on less powerful devices.

2.4 Running the Model: Cloud vs Local

A common approach to ML at the edge is to process and train data at data centers or cloud servers. Cloud computing has allowed for devices that are otherwise restricted in power to gain access to higher level capabilities, granted that the device has internet access[10]. In addition to internet requirement, cloud computing imposes additional constraints. One must factor in the latency loss involved with relying on a cloud, the cost of keeping cloud servers running and the edge device connected, and the risks of having potentially private data transferred over the network. In a world where network capacity can often be overloaded and speed can vary from location to location, cloud computing is not a feasible method of granting access to models for every device. Distance inherently impacts the latency and connection between cloud and device, with the optimal solution often being the deployment of additional data centres or networks near the device when far from the host machine[16]. In situations where internet access is consistent or latency isn't a worry, cloud computing is the best way to ensure accuracy and model strength. However, frameworks and methods have been achieved in order to allow edge devices to collect and train data locally while still maintaining performance similar to that of models running on the cloud. There are two methods in particular that offer extreme potential for ML at the edge, federated learning and the Edge2Train framework.

2.5 Federated Learning

Federated learning can be considered a hybrid between fully centralized cloud computing and fully local edge training. Federated learning alleviates some of the aforementioned issues that arise with cloud computing, while gaining access to some new benefits. At a baseline, federated learning is primarily dedicated to using edge devices as a resource in order to

uniquely train and personalize data across locations after being given access to the global ML model from a central server. It is then possible for said devices to push their model changes to the cloud, allowing the global model to be updated[7]. This decentralized method of data capturing and training allow for far more specialized and developed models to form. In fact, federated learning often allows for higher levels of accuracy than found in models run on solely the cloud.

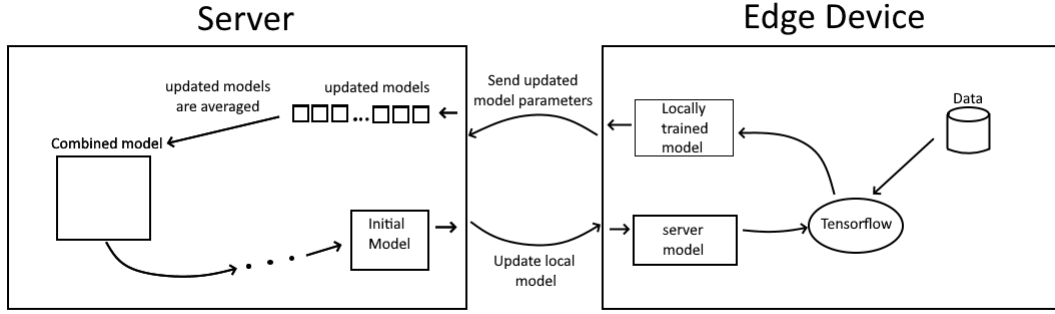


Figure 1: Diagram of Federated Learning

In an experiment run by the Nagoya Institute of Technology[2], simulations were done in order to compare centralized ML and ML with federated learning. Using the MNIST and CIFAR-10 datasets, the researchers found that centralized ML achieved 65/73% accuracy with MNIST and 54/62% accuracy with CIFAR-10, whereas federated learning models showcased accuracy levels of 92/97% and 86/94% on each respective model. At worst, federated learning’s accuracy was 24% higher than centralized ML, and at best an increase of 32% was seen. These results show that the advantages of federated learning are much greater than that of centralized ML, while also allowing for offline functionality.

The primary limitation of federated learning when it comes to edge devices is that it is only capable on systems with powerful processors. While this allows for the majority of modern smartphone devices and smart cars to take advantage of it, certain edge devices such as security cameras or smart appliances may be unable to utilize this approach.

2.6 Edge2Train

Not every edge device comes with the resources necessary to take advantage of federated learning, yet would likely still prefer an alternative to running on the cloud. For such devices, the Edge2Train framework is an alternative with a large amount of potential. Edge2Train is a novel framework developed in order to allow small micro-controller units (MCUs), which are often limited in memory and computational power, to gain the ability to train models locally and offline. The framework is able to learn from the edge device’s sensors in order to convert that data into something the model can understand, train that data on the model in the MCU, and then output that data into commands that can interact with the edge device’s application, allowing for actuation. The Edge2Train framework then monitors the data it gains as the application runs, training and updating the model [15].

Edge2Train uses C++ in order to optimize training methods to function on less powerful devices, meaning the general structure for model creation differs compared to development on more resource-capable applications. Edge2Train’s training methods optimize data to be more compatible with these lesser-powered edge devices, consuming less resources while maintaining scalability. The MCUs used in the experiment were highly limited in terms of memory, with the range of SRAM going from a minimum of 20kB to a maximum of 520kB. Despite this limited SRAM, all models were capable of being trained on the MCUs, however larger datasets are subject to being done in batches in order to avoid overflow[15]. Simulations also showcased that accuracy is relatively maintained despite the loss in resources when compared to models trained on computers. The first dataset in the experiment showcased 96.67% accuracy on the computer and 90.00% on the Edge2Train device. The second dataset actually indicated higher accuracy on the edge device than on the computer, with Edge2Train boasting 92.85% accuracy, whereas the computer had 91.66%. These results indicate that Edge2Train’s framework is capable of producing classifications at the same level as those on models trained on powerful computers, without the requirement of needing a cloud connection.

2.7 Framework Limitations

While frameworks such as Edge2Train and Federated Learning offer a great deal of potential for ML at the edge, they are not always available options for every project. Edge2Train assumes the device is using a sensor in order to collect data, which is not applicable to every edge device. Similarly, Federated Learning was previously referred to as a framework that requires powerful processing, meaning it is mostly restricted to more modern devices, such as smartphones. In cases where frameworks such as the aforementioned are unavailable, whether due to resource restrictions or device limitations,

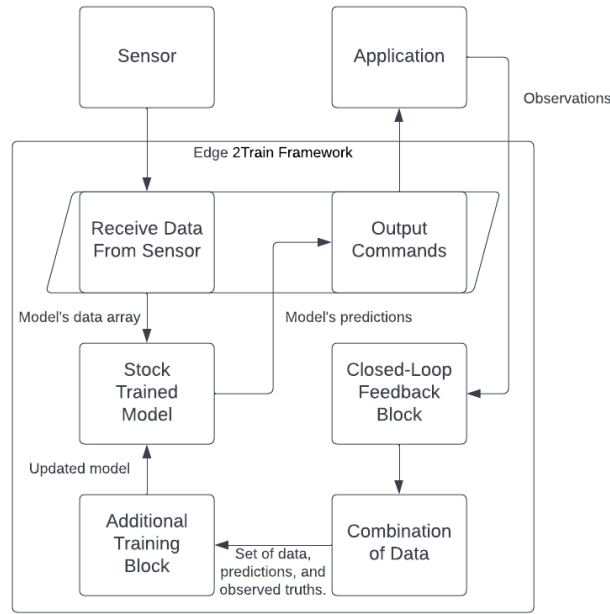


Figure 2: Diagram of Edge2Train

cloud computing is still a viable option. Not every project requires minimal latency, and latency can be mitigated through smart usage of networking and data centre structure[16]. ML at the edge requires careful consideration of the project's requirements and understanding of the device's capabilities. No two projects are likely to have the same options available to them due to the wide variety of edge devices and their various limitations and resources.

2.8 Data Compression

Data compression is a typical technique for reducing data size and improving transmission efficiency[8]. In order to shrink the amount of data processed for edge devices, developers have tried to minimize the data by deploying different ways of data compression. These ways are classified into lossless data compression methods and lossy data compression methods. Lossless data compression methods essentially depict the loss of accuracy in raw data when compressing since it is supposed to restructure the raw data perfectly, therefore, lower compression ratio is lower. Lossy data compression method is where redundant data are eliminated during data compression, and higher CR can be achieved.

2.9 Spatiotemporal Data Compression

Spatiotemporal data compression is one method of data compression that can be used when dealing with large amounts of data. Spatiotemporal data compression relies on clustering data together in order to reduce the amount of data that needs to be processed. This is done by looking at similar series of data and grouping them together based on their spatiotemporal similarity. For example, in a sample of 100 data series, series 1 and 2 represent similar results. In this case, series 1 can be used to represent series 2, or series 2 can be used to represent series 1. Instead of counting them separately, they can be grouped into the same category, reducing the amount of data that needs to be processed. This is a standard example of clustering, however spatiotemporally clustering data is more complex. In order to cluster data spatiotemporally, the algorithm works using data aggregation as opposed to data regression. By applying higher weight to newer data, the algorithm is able to more accurately group the data into clusters without having to constantly recluster datasets[19]. The algorithm will also check to see when clustering is no longer worth the cost, at which point it will stop clustering and begin using an ordered compression approach. Order compression is similar to standard data compression models, checking the remaining data to see if it can be compressed further. If data sets 1-7 were clustered, then the algorithm will check to see if data sets 8-10 can be compressed further. If 8 and 9 show no change, then it is likely that 10 will also not require further compression. This allows the algorithm to return the information that 8 and 9 were left unchanged back to the cluster head, ignoring 10, as it can be assumed no difference will be found.

While an incredibly useful tool, spatiotemporal data compression is not without its limitations. The algorithm is designed to be used with cloud computing in mind, meaning any edge devices that operate without access to the cloud cannot leverage the method. Processing must be done on both the edge device and the cloud servers at least once each,

with further iterations also having to be considered. In addition, the algorithm is unlikely to yield significant results when dealing with small datasets, as the algorithm is designed to work with big data. As such, for edge devices operating on smaller scopes, there's likely to be little to no benefit from using spatiotemporal data compression.

2.10 Model Compression

Model compression is very significant when it comes to edge devices, it is the strategy to create smaller and thinner models that allow for more computation and energy efficient or negligible or even no loss on accuracy of the data. Model compression aims to lighten the model, improve energy efficiency, and speed up the inference on resource-constraint edge devices, without lowering the accuracy[17]. Due to the limited memory and compute power of edge-devices, it is important to tailor the model so it can fit and execute efficiently on those devices. Therefore, techniques have also been developed for compression ML models to make the models lighter and faster and to aid their deployment on the edge[11]. There are two broad approaches for doing this (a) quantization, which reduces the precision with which parameter values are stored, which in turn lowers the memory footprint and therefore potentially makes computations faster, and, (b) model pruning, which reduces the number of model parameters and therefore improves storage and compute time[11].

2.11 Quantization

Quantization is a form of model compression that can be utilized for edge devices. As compression helps lighten the model, quantization reduces the accuracy in which values of parameters are saved causing a lower memory footprint, therefore theoretically making computations at a quicker pace. Conventional quantization techniques are applied to entire network layers or on kernel weights without considering the dynamic properties of the feature map. Such quantization sometimes hurts the overall performance of a deep learning model despite large reductions of energy usage[11]. Due to the loss of memory footprint, the performance and precision decrease, which in return does not allow for reliable compression on a model to be properly trained.

2.12 Model Pruning

Model Pruning is also another model compression that can be utilized for edge devices. A very similar form to Quantization, Model pruning, reduces the number of model parameters and therefore improves storage and the time it takes for computations as well. Even though this form of model compression displays the great potential to make an enhanced and faster model, pruned edge-trained models often have lower accuracy, and therefore designing power-efficient algorithms for training neural networks on edge devices is an active research area[11]. Considering both ways of model compression, the direct effect is an overall decline in energy use which is a key component for a fast, reliable edge device to be trained. However, the limitations of knowledge and research on the current model compression methods raise a big issue due the loss of accuracy and precision.

2.13 Libraries for ML at the Edge

In order to actually facilitate ML at the edge, it is important to have libraries that can be used to implement the machine learning while being scalable and efficient. Two popular libraries for ML currently are TensorFlow and PyTorch, each having mobile versions, TensorFlow Lite and PyTorch Mobile respectively. Both offer a number of pre-trained models that can be used for a variety of tasks natively, allowing for easy implementation of ML for edge devices. These models are light versions of their original counterparts, with TensorFlow Lite, for example, being less than 300KB in size when used exclusively for image classification and 1MB at max when all supported operators are used[5]. While PyTorch Mobile offers no official numbers on sizes, it also supports the ability to customize the models used in order to reduce size.

The two libraries are very similar in terms of optimization, benchmarking, and CPU acceleration. The main difference between the two comes down to personal preference and the type of project being worked on. In cases where a program is meant to be run on desktop devices and edge devices, PyTorch Mobile may be the better option. This is due to the conversion framework of code from PyTorch to PyTorch Mobile being more straightforward than the conversion from TensorFlow to TensorFlow Lite. PyTorch Mobile runs on the same codebase as PyTorch, meaning that any model training done on PyTorch can be converted to PyTorch Mobile by running the PyTorch Mobile Lite Interpreter. TensorFlow Lite, on the other hand, is an entirely separate framework from TensorFlow. Some operators available on TensorFlow cannot be used on TensorFlow Lite, meaning that one must be aware of what operators they are using if they plan on using the TFLite converter[9]. Both libraries are worth exploring when beginning development for ML at the edge, as the mileage gained from each library will vary depending on the project.

3 Security

Machine learning technology has made great strides lately and is widely operated in numerous fields including image classification, speech recognition, etc. However, these models are also vulnerable to various attacks that may compromise the security of its application systems. Machine learning on Edge devices is a newer technology and different from Machine Learning in general. Edge ML is an approach that allows smart devices to process data locally either at the device level or via local servers using machine learning and deep learning algorithms, minimizing their need on cloud networks. The word "edge" refers to processes carried out by deep learning and machine-learning algorithms at the device- or local-level. Machine Learning. Unfortunately Edge devices are not without their fair share of security threats and exploits.

Security threats to Edge devices

- DDoS attacks - Distributed Denial of Service also known as DDoS attacks have always plagued the IT industry and big businesses (e-commerce). In simple terms, this attack's purpose is to prevent a web service to function normally making it inaccessible to the users. Edge computing networks can also be a victim of DDoS attacks. An edge device is not as powerful as cloud servers because of its limited resources, thus this attack is aimed to deplete the resources of an edge device by sending a large number of packets to its the system causing it to slow down and crash eventually. DDoS attacks can be categorized into UDP flooding attacks, ICMP flooding, SYN flooding, ping of death, and HTTP flooding. [14] [1]

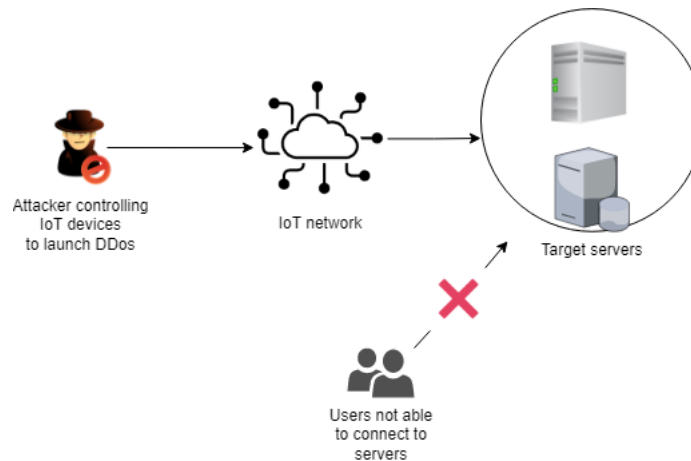


Figure 3: DDoS attack

- Malware Injection - A malicious service or code is introduced into the system during this attack. The attacker manipulates the Edge Device into thinking that the newly implemented service is an instance of the system as a whole. The attacker's code is then run after a legitimate user request is forwarded to this service. Trojans, worms, and other types of malware cause data breaches, power drains, and performance degradation. [14] [1]
- Eavesdropping, spoofing, MITM attacks - Eavesdropping attacks use listening and snooping techniques on the networks to gather data from end nodes or locate them. As soon as an attacker has access to vital data like a node's configuration and identification, the integrity of systems is compromised. In spoofing attacks, In the Edge network, a spoofing node impersonates a genuine device by using a fake medium access control (MAC) address or RFID tag. MITM attacks also known as Man in the Middle attacks are when an attacker gains control over the Edge Network and uses jamming signals to watch and listen(eavesdrop), alter communication going between the Edge devices. [14] [1]

Defenses for Security Threats

- DDoS Attacks - Standard defenses against DDoS attacks involve filtering network traffic and redirecting traffic that has been deemed malicious. This harmful traffic is funneled into resources that function to effectively contain it so as to mitigate the harm caused. More complex machine learning assisted measures have been developed in order to identify DDoS attack traffic. Doshi et al. [4] found that machine learning-assisted algorithms are adept at minimizing false positives which is an important consideration so as to not hinder legitimate network traffic from being able to access necessary services. They also tested several different machine learning-based algorithms for the purpose of DDoS detection and concluded that the random forest, KNN, and neural network classifiers were the most successful, achieving an accuracy score of 99%.

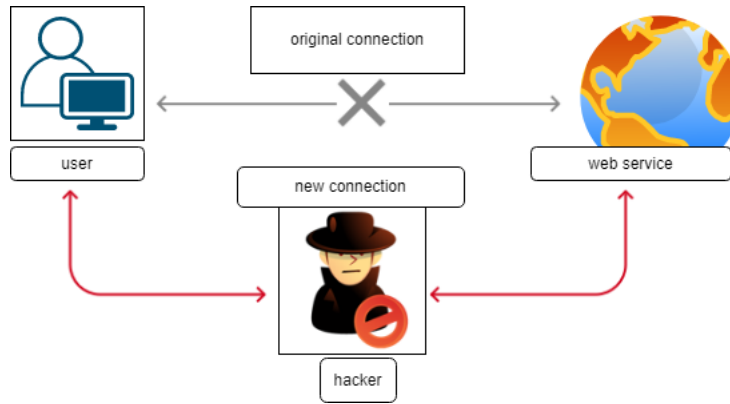


Figure 4: Eavesdropping

- **Malware Injection** - A survey was carried out by Singh et al. [14] in which they reviewed the available literature surrounding ML-based defenses against malware injection attacks, among a series of other attacks that plague edge computing systems. When the available ML-based techniques were assessed, the strongest performance came from a linear support vector machine (SVM) based classifier method that was tested against Android attacks. This method showed 99.2% precision, but does need to be tested on larger datasets in the future.
- **Eavesdropping and Spoofing** - Data encryption is a vital tool in helping to prevent eavesdropping attacks. Additionally, physical security measures can help protect against physical methods that an attacker might use such as placing bugs on victim devices. Hoang et al. [6] developed a way to take wireless signals and construct structured datasets out of them in order to then utilize SVM classifiers to aid in detection of eavesdropping attacks at the physical layer. In terms of dealing with spoofing attacks, in order to prevent or mitigate the damage from these attacks it is important to take measures in authentication as well as packet filtering. Packet filtering involves controlling which IP packets are allowed to access a network interface based on whether the source address is deemed legitimate or safe.

Ethical/Legal Constraints

- Many technical privacy challenges in IoT remain unsolved despite numerous efforts. Internet of Things require legal frameworks that can evidently depict the limitations, rights, liabilities, and duties of each system participant. Data is vital, In order for us to make reliable estimates and decisions. Without adequate and consistent data, it is hard to accurately assess the problem and reach to a conclusion. IoT gathers a plethora of different types of data which including but not limited to: Purpose Data, Hardware Data, Limited Disclosure Data, Network data, Traffic data, meta data and much more. These are useful data sources that aid Machine Learning professionals in designing models that use ML to protect privacy more effectively for which, high-quality data is the key. However, because of the features and properties of the IoT ecosystem, access to good quality data is not straightforwardly attainable. Compatibility between various devices is among the key problems. IoT hardware is made with a variety of standards and technologies. The main concern that has a significant impact on data collection and usage is legal policies.
- Each of the aforementioned data sources has an owner who has his/her preferences and necessary procedures for data storage and use. All rules pertaining to the owners and governing parties should be taken into account before using any data. As a result, It is not possible to utilize the full potential of Machine Learning on Edge devices in privacy protection solutions and services without an systematic and thorough regulatory framework. Manufacturers should clearly state how they use and disclose data subjects' information, and this requires the enforcement of particular laws and policies. The legal duties for data leaks are another concern that is yet not well understood. Data in various IoT layers is accessible to a wide range of parties.
- IoT has different layer architectures which include: Perception Layer, Network Layer, Application Layer. The application layer is essential for data storage and processing functionality. Data centers are extremely alluring target for cybercriminals. Authentication and access control are the biggest privacy challenges for data storage operations. As a result, guarding them against wrong doings requires sizable investments and efforts.

Ethical/Legal Constraints

- Data governance can make use of ML methods to regulate IoT system access rights. Learning models can identify suspicious requests in authentication and access control procedures based on the data that are available from prior

access behaviors. Xiao et al. (2016) proposed a wireless network authentication method that employs radio channel data to provide physical layer authentication. To pick the optimum threshold values for radio channel data that can be categorized as public data, the authors employed two reinforcement learning techniques, Dyna-Q and Q-learning. Aissam OUTCHAKOUCHT a PhD Student, Laboratory LISER at IPI, Paris, France integrated ML and blockchain technology and proposed a reinforcement learning methodology that is developed to assist smart contracts in managing control decisions dynamically depending on feedback from the environment. Bitra Darvish Rouhani, M. Sadeh Riazi, Farinaz Koushanfar and proposed a framework that can maintain the privacy of both the learning parameters and the input data. The garbled circuit, a cryptographic technique based on logical synthetic

Open Source Projects Aimed at Securing ML Systems

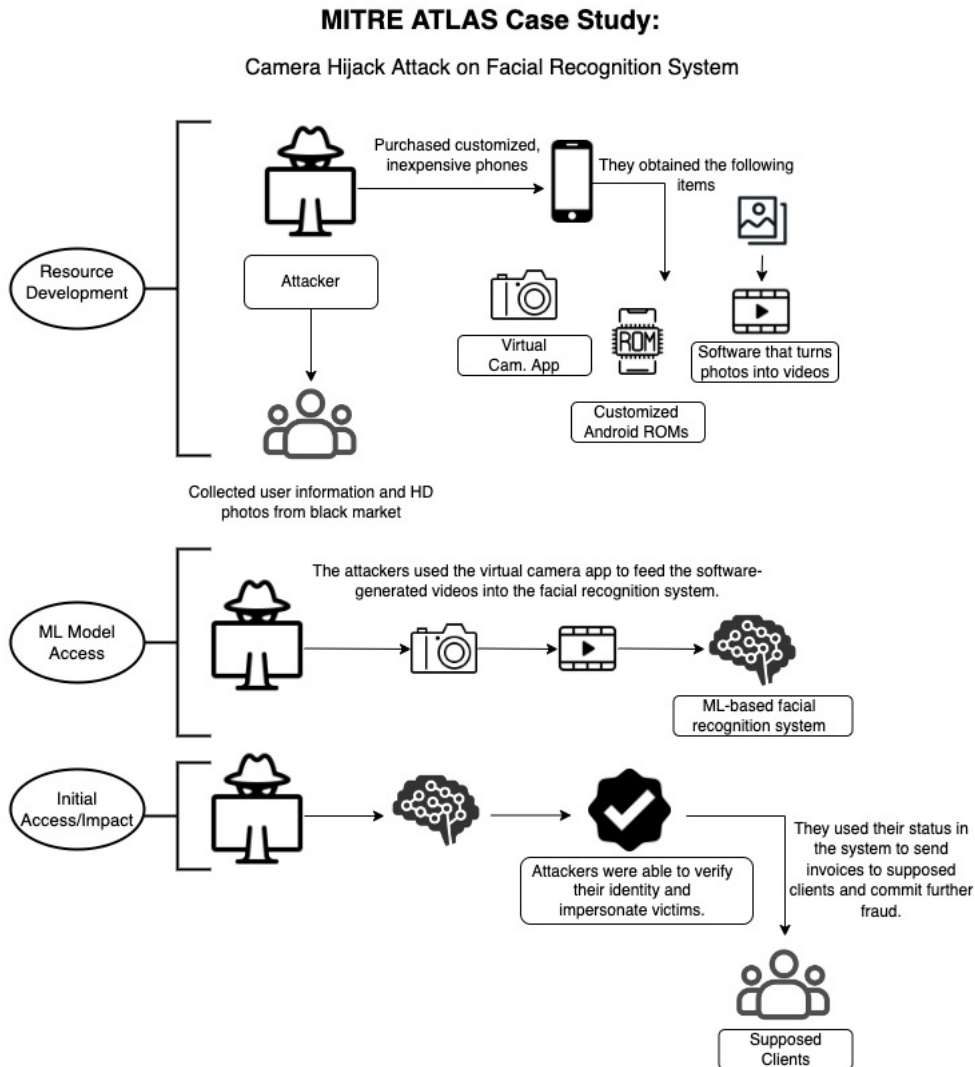


Figure 5: MITRE ATLAS CASE STUDY

- IBM's Adversarial Robustness Toolbox - The Adversarial Robustness Toolbox (ART) is hosted by the Linux Foundation AI & Data. It was created by IBM and then donated to the foundation. It tackles the threats of evasion, poisoning, and inference. An evasion attack modifies the model inputs and changes the behavior, while a poisoning attack tampers with the training data. Inference attacks deal with leaking data when an attacker has been able to infer it. It currently supports many popular ML frameworks such as TensorFlow, PyTorch, Keras, etc. and aims to support more in the future [12].
- Microsoft's Counterfeit - Counterfeit is a command-line interface developed by Azure to assess the security of ML systems. It comes preloaded with attack algorithms and makes them accessible to security professionals. It allows these experts the ability to simulate attacks against ML systems. It can analyze models that are hosted in various

environments from cloud environments to edge environments. Counterfeit also supports many data types such as text, images, etc.

- EvadeML - EvadeML is a project started by PhD student Weilin Xu at the University of Virginia. The project's goals are as follows: provide ready-to-use datasets, models that are pre-trained and are ready to simulate attacks, as well as providing currently existing methods of attack. The project also covers a way to visualize adversarial examples and provides current, state-of-the-art defense methods [18].
- RobustML - RobustML is a robustness resource that has a wide variety of defenses available to peruse. These defenses also have published code to view along with research papers. The defenses listed have open-source code as well as pre-trained models. For each defense, RobustML lists the dataset, the threat model, the natural accuracy, the claims, and the analyses.
- TensorFlow's Cleverhans - Cleverhans is a Python library that gauges different ML models' weaknesses against adversarial attacks. It supports the JAX, PyTorch, and TF2 frameworks. The library also provides the implementations of these adversarial attacks as references. It also includes defense implementations that can be used to preserve ML models against malicious attacks [13].
- AdverTorch - AdverTorch is a robustness toolbox built at the Borealis AI research lab. It revolves around the PyTorch framework and it includes ways to generate adversarial attacks as well as ways to defend against these attacks. The defenses included in this toolbox are divided into two categories: pre-processing defenses and robust training methods [3].

4 Deployment

Deployment overview

When developing an application there are certain steps that must be taken to ensure the successful outroll of a new application. When developing there are a few great best practices that are used throughout the industry. It starts with knowing the requirements, building a design, development, testing, deployment, and maintenance. This is the general cycle the industry uses for new applications. The first stage is getting the requirements needed from whoever is making the request. This includes the usage of the apps if there are any restrictions and how they need it to be used. Once all the requirements are accepted the team would create a design and give it back to the requester and if the design is acceptable they then move onto the deployment stage. This is where all the things in the design step get built all while keeping communication to the requester. Once development is complete and approved the testing phase begins. Making sure the application works as designed and making sure there are now security issues that were missed in development. After these stages are complete then comes the deployment phase. This stage is key because it is where the application is being.

Deployment types

There are a few deployment types that are regularly used throughout the industry. Having it hosted on the cloud, on premises, and a hybrid approach. The first being hosted on the cloud would be using servers that are provided by a third party company and use a pay as you go approach. When having it deployed in the cloud you have all the resources and benefits the cloud has to offer. Such has nearly endless compute power and storage as well as the elasticity of having all the resources on demand.

Cloud Deployment A fully cloud deployment entails the application being fully hosted on a cloud provider. The benefits that come with being on the cloud are the elasticity and the lower upfront cost to get the application deployed. The cloud has many tools and resources that assist in reaching edge devices and are scalable to deal with the increased activity. Each cloud platform has in house tools as well as third party softwares that can integrate with applications. Since we are planning on deploying a machine learning model the compute power of the cloud can decrease the time it would take to get results. The cons of a fully cloud approach is that in some of the use cases of the application having access to the internet is not possible and would render it useless as of now.

On Premise Deployment: With an on Premise Deployment the upfront cost of the servers to deploy the application would be higher. The On premise approach would be good for government requirements where certain information can not be stored on a shared server. This approach has been used the longest but there has been a shift to cloud infrastructure. A lot of the same issues arise here if the edge device can not reach the internet the application would be useless.

Hybrid Deployment: This approach combines both having parietal applications within the cloud as well as parts on premise. So the benefit is if you have an application deployed on premise and it has a function that requires a lot of compute power that task can be outsourced to the cloud to have that task be finished sooner.

Deployment on edge devices

An edge device is a device that provides a user to gain access to an application. For example if there is an application that is hosted through an app store your phone or any other device that can access that application would be considered an edge device. Now when building the application there will be certain things to keep in mind. So the different types of edge devices that will be used. In our project the edge devices will be computers, watches, and military equipment. You must design the application with these edge devices in mind. Because the phone based application should look different from the laptop based version.

There are many ways to deploy this application to edge devices. Some of the options include through the internet whether it be through the app store, website, as well as programming the application on to the devices. Once the application is ready and can be containerized it can be sent and installed to the edge devices.

Internet: As discussed earlier the application should be able to run without internet so the development team of the application has to ensure when developing the application that the edge device resources are kept in mind. The compute power of a smartwatch versus a computer or laptop are vastly different. This will have to be kept in mind since the resources of the device will be used in some applications. When the application does have access to the internet then the resources of the cloud can be used to have a faster compute time. As well as pushing updates to the application and being able to see if the model is accurate.

Dashboards

To monitor changes in the data (data drifts) due to x factors, we can compare and contrast the distribution of the training data with the data our ml model finds while in production. The process of monitoring the model starts once we are in production, and to ease this process and automate it to an extent, we can use tools such as Boxkite and Evidently that generate periodic reports from our data in the form of a html page or as a json.

To create a useful and effective model, a precise optimization and trimming approach must be used with scientific libraries (like the ones mentioned above). When it comes to model deployment on edge devices that need is further exacerbated.

Similar to process optimization, the goal of ML model optimization when carried out in a resource-constrained setting is to achieve effective run-time performance. The procedure entails employing a variety of optimization techniques and algorithms to make incremental, quantifiable improvements.

Techniques that could be applied:

Pruning: Refers to a group of methods for reducing the size of a network in order to enhance computational performance and occasionally resolution performance. By identifying the nodes that, if removed from the network, would not significantly influence network performance, these strategies aim to remove nodes from the network during training Regularization: The performance of the model can be significantly improved by avoiding overfitting. Regularizations of type L1 and L2 (weight decay) are the most popular. These add a new term known as the regularization term and update the general cost function. Concretely; Quantization: By using fewer bits for model weights and activations, quantization approaches can increase inference speed and are particularly effective when employed during training.

Different model types

Edge devices are considered a fundamental part of distributed AI systems. Back in the day, technological devices were only able to manifest data by sending it to a cloud for further analysis. However, this has changed as now those very own IoT devices can perform their computations thanks to the major improvements we have made with software devices and ultimately resulting in edge computing. Edge computing has revolutionized how data is processed from billions of devices all over the world.

Monitoring

Through the deployment process we have to utilize applications to help aid the monitoring of it. There are certain things that we need to monitor within the app such as health checks, seeing how much the app is being used and if there are any accuracy errors with the application. There are a lot of monitoring software choices on the market but these three would best suit our needs within this deployment. The first software monitoring tool we will be discussing would be Datadog. Datadog is a company that specializes in observability for applications whether it be on-premise or cloud based. It “uses a Go-based agent and its backend is made from Apache Cassandra, PostgreSQL and Kafka”[2]. By using these tools and working with the data dog team you can have full insight into the app and get scheduled metrics and logs reported. The next possible solution that could be used is Splunk. Splunk is another monitoring software that specializes in searching through large amounts of data. Some features it has are being able to set alerts, generate reports and visualizations that give an easier insight to shareholders. The final option that we will discuss will be prometheus. Prometheus is an open source monitoring software that has tools built in that DevOps teams can utilize to create a monitoring system that

works for their specific needs. One of the benefits of Prometheus is its open source meaning that it is a free software that is community built and updated. Which would save a lot of cost compared to our first two choices. It also utilizes Kubernetes which would be used in the deployment. So having prometheus as a monitoring service that already has kubernetes integrated into it would be the best practice.

Since this application has to work with or without the internet we have to add a function that stores information that needs to be relayed back to the monitoring system. Once the device connects back to the internet then it can be relayed back to the monitoring software where the team can look at the data. The reason monitoring is important is to make sure that the machine learning model is being used to its full potential and reducing type II errors. You can also monitor to see how many people are using the application to know if it is useful. Having a monitoring system that a data analyst or scientist can utilize that will allow insight and recommendations on the use of the application.

5 Conclusion

In conclusion, machine learning on edge devices is a subject that has seen much discussion recently. While there are many challenges to overcome, from working with resource constraints to ensuring the security of the data, there have been advances in the field through new frameworks, security protocols, monitoring tools meant primarily for ML at the edge. With all this in mind, we believe that the future of ML at the edge is bright, and that we should soon be able to begin development of an application demonstrating the benefits of ML at the edge.

6 Backlog Content

In this section we keep ideas that we are still developing but are not ready to be part of the paper yet...

6.1 Development Backlog

One aspect that received research but no writing is application development. This is a subject that relies on the specific application and edge device chosen. As such, the topic cannot be adequately explored without understanding the more minute aspects of developing ML on the edge. With the current state of the project and information gathered, application development should be a primary focus of future work. Understanding what sort of algorithms are available for different edge devices is a guiding principle for the development of the application, so with the knowledge of general ML procedures and specific strategies for edge devices, the process of developing an application should be feasible.

Model packaging is also an important aspect to consider. Some brief research was done into MLOps tools, but more research is needed in regards to web based frameworks, such as when to use MLOps tools over web based frameworks, how to get each running on edge devices specifically, and the benefits and drawbacks of each. It's highly possible that the options here tie into the decisions made in the application development process, and as such, the two should be considered together when doing future research.

6.2 Deployment Backlog

For the future of this project we would recommend doing more research and starting basic tests on a sample device. So having a smartwatch or phone and deploying a sample application to it and testing to see what issues arise. As well as having the monitoring system in place to see how it performs with the internet and without the internet. Seeing that these applications will have to work with and without the internet there needs to be a way to have that data stored until the internet is available. As well as being able to push updates to said application. We have a layout for the different aspects dealing with the deployment now we just need the testing to be done. There are many ways to deploy this application and find the most efficient when looking at cost performance and scalability. A deployment for a small team compared to an entire enterprise will vary just due to the fact that some applications may have a bottleneck which can slow the application down when it comes time to scale.

References

- [1] Mohammad S Ansari, Saeed H Alsamhi, Yuansong Qiao, Yuhang Ye, and Brian Lee. Security of distributed intelligence in edge computing: Threats and countermeasures. In *The cloud-to-thing continuum*, pages 95–122. Palgrave Macmillan, Cham, 2020.
- [2] Muhammad Asad, Ahmed Moustafa, and Takayuki Ito. Federated learning versus classical machine learning: A convergence comparison, 2021.

- [3] Gavin Weiguang Ding, Luyu Wang, and Xiaomeng Jin. AdverTorch v0.1: An adversarial robustness toolbox based on pytorch. *arXiv preprint arXiv:1902.07623*, 2019.
- [4] Rohan Doshi, Noah Apthorpe, and Nick Feamster. Machine learning ddos detection for consumer internet of things devices. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 29–35. IEEE, 2018.
- [5] Google. Tensorflow lite.
- [6] Tiep M Hoang, Trung Q Duong, Hoang Duong Tuan, Sangarapillai Lambotharan, and Lajos Hanzo. Physical layer security: Detection of active eavesdropping attacks by support vector machines. *IEEE Access*, 9:31595–31607, 2021.
- [7] Hsin-I Hsu. Training ml models at the edge with federated learning, Jun 2021.
- [8] Alicja Kwasniewska, Maciej Szankin, Mateusz Ozga, Jason Wolfe, Arun Das, Adam Zajac, Jacek Ruminski, and Paul Rad. Deep learning optimization for edge devices: Analysis of training quantization parameters. In *IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society*, volume 1, pages 96–101, 2019.
- [9] Dhruv Matani. On-device deep learning: Pytorch mobile and tensorflow lite.
- [10] Peter Mell and Tim Grance. The nist definition of cloud computing, Sep 2011.
- [11] M. G. Sarwar Murshed, Christopher Murphy, Daqing Hou, Nazar Khan, Ganesh Ananthanarayanan, and Faraz Hussain. Machine learning at the network edge: A survey. *ACM Comput. Surv.*, 54(8), oct 2021.
- [12] Maria-Irina Nicolae, Mathieu Sinn, Minh Ngoc Tran, Beat Buesser, Ambrish Rawat, Martin Wistuba, Valentina Zantedeschi, Nathalie Baracaldo, Bryant Chen, Heiko Ludwig, Ian Molloy, and Ben Edwards. Adversarial robustness toolbox v1.2.0. *CoRR*, 1807.01069, 2018.
- [13] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, and Rujun Long. Technical report on the cleverhans v2.1.0 adversarial examples library. *arXiv preprint arXiv:1610.00768*, 2018.
- [14] Shivani Singh, Razia Sulthana, Tanvi Shewale, Vinay Chamola, Abderrahim Benslimane, and Biplab Sikdar. Machine-learning-assisted security and privacy provisioning for edge computing: A survey. *IEEE Internet of Things Journal*, 9(1):236–260, 2021.
- [15] Bharath Sudharsan, John G. Breslin, and Muhammad Intizar Ali. Edge2train: A framework to train machine learning models (svms) on resource-constrained iot edge devices. In *Proceedings of the 10th International Conference on the Internet of Things, IoT '20*, New York, NY, USA, 2020. Association for Computing Machinery.
- [16] Thang Le Duc Umeå University, Thang Le Duc, Umeå University, Rafael García Leiva IMDEA Networks Institute, Rafael García Leiva, IMDEA Networks Institute, Paolo Casari IMDEA Networks Institute, Paolo Casari, Per-Olov Östberg Umeå University, Per-Olov Östberg, and et al. Machine learning methods for reliable resource provisioning in edge-cloud computing: A survey: *Acm computing surveys: Vol 52, no 5*, Sep 2020.
- [17] Dianlei Xu, Tong Li, Yong Li, Xiang Su, Sasu Tarkoma, Tao Jiang, Jon Crowcroft, and Pan Hui. Edge intelligence: Architectures, challenges, and applications, 2020.
- [18] Weilin Xu, David Evans, and Yanjun Qi. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. In *Proceedings of the 2018 Network and Distributed Systems Security Symposium (NDSS)*, 2018.
- [19] Chi Yang, Xuyun Zhang, Changmin Zhong, Chang Liu, Jian Pei, Kotagiri Ramamohanarao, and Jinjun Chen. A spatiotemporal compression based approach for efficient big data processing on cloud. *Journal of Computer and System Sciences*, 80(8):1563–1583, 2014. Special Issue on Theory and Applications in Parallel and Distributed Computing Systems.