tinyML Deployment Working Group White Paper #1

February 20, 2023

There is far more than "fit & predict" development required to deliver Tiny ML based products.

This is the first white paper in a series exploring challenges and solutions for deploying ultra-low power machine learning (ML) at the edge of the cloud. The authors are members of the tinyML® Foundation Deployment Working Group. The opinions expressed are not necessarily representative of the tinyML Foundation, its sponsors, or the authors' employers.

Motivation

The digital transformation market is growing rapidly and is set to reach a volume of one trillion dollars by 2025. Machine learning is also quickly gaining traction and is projected to grow to volumes amounting to trillions of dollars over the next couple of decades. Such market dynamics are resulting in an immense demand for appropriate solutions, and the development of a supply to match.

Even though compute resources for machine learning training have been doubling every six months for the past ten years, lack of data is one of the main reasons why ML projects fail. This problem is especially significant for the "offline businesses", where ML algorithms need smart sensors to gain access to the real world. Basically, everything around was digitalized except the offline infrastructure that is built around the human population.

The availability of data is often overestimated by companies, leading to a breakdown in planned processes that require better datasets to support operations. Therefore, hardware is taking up a bigger role in software-related processes, as machine learning algorithms and digital transformation software need smart sensors to gain access to the real world. The availability of this vital link, and the overseers ensuring the authenticity of the data, is the key to providing constant data streams for improving machine learning efficiency.

Significant technical and commercial progress has been made by the tinyML community towards both hardware and software optimizations to make evaluation and training of the ML models directly on sensor both fast and energy efficient. There are even commercially successful “no code” solution platforms that allow the development of models without the user having to write code.

Unfortunately, the scope for most of the tools is focused on the development rather than deployment process and bounded by ML-model training and loading the resulting model onto the selected target device. What is missing is the relevant support for device deployment and management processes, which are an essential part of any production-ready solution.

An apt analogy is that nice cars have been developed (algorithms and tools to build models) but no freeway system has been appropriately defined (deployment) to run these cars on. This is not a challenge that is unique to Tiny ML but is also faced by Internet of Things (IoT) devices. As such, we intend to leverage existing work to deploy the IoT and focus on the unique requirements and features of Tiny ML.

The main goal of this paper is to start a discussion about deployment best practices and gain some consensus or identify divergent opinions from within our community. Then we wish to move outward to those who should be interested in deployment for either their product development or application. For these people, we hope to provide a roadmap that they will choose to support or will leverage the knowledge into their own efforts. Lastly, we wish to engage with the wider community of ML users and developers.

Deployment Definition

Deployment is all the activities after services are ready (cloud side); the code and model are ready; and the physical devices are built but not yet programmed or configured before shipment or installation in the field.

This paper addresses Tiny ML deployments. It is important to realize the impact that these deployments can have on society. It is important to ensure that Tiny ML deployments are safe, trustworthy and ethical. Some of the issues related to Tiny ML deployments and the actions taken based on these deployments are: fairness and inclusiveness; reliability and safety; privacy and security; transparency; and accountability.

IoT Deployment Fundamentals

Before discussing the specifics of Tiny ML deployments, it is important to understand IoT fundamentals which are common to all deployments.

When planning an Internet of Things (IoT) deployment, there are various technical and operational aspects to consider. Here, we will cover some of the key points that apply to most IoT deployments which will also be relevant to deployments of Tiny ML-based devices.

IoT deployments center around multiple devices or device fleets that are deployed in the field, which can include end-consumer products, commercial building installations, or smart factory integrations. The devices in these fleets will each connect to a cloud service backend either directly or through one or more gateways. The device software and connectivity must align with the cloud configuration and security in order for an IoT deployment to operate successfully at scale. Unlike typical cloud-native infrastructure deployments where a DevOps team can spin up compute resources on-the-fly or execute processes that dynamically allocate services, IoT deployments include real-world devices outside the datacenter that are not so ephemeral. With this in mind, we will discuss various considerations for IoT deployments below.

Device Connectivity

The edge devices in an IoT deployment can connect to cloud services in a number of ways. Connectivity at the physical (PHY) layer can include: Wifi, Bluetooth, ethernet, LoRa, IEEE 802.15.4 (Zigbee or Thread), or even cellular. Often, choices are made based on the cost of the radio or chip, the networking range requirements, and the power consumption of the device – battery vs. continuous power. Also, the network topology of the expected edge environment is an important consideration. Will the device connect directly to the internet through a wifi router, paired to a mobile device with Bluetooth via an app, through a commercial IoT gateway, from a firewalled enterprise network, over a LoRaWAN network in a remote area, or through a Zigbee or Thread border router as part of a mesh network? All of these examples will exhibit their own unique mechanisms for connectivity based on the network. Finally, once the device is connected, there are a number of protocols for data exchange to consider. Typical IoT standards include: MQTT, HTTP, Websocket, and CoAP. Each of these protocols have their strengths and nuances compared to the others.

Device Provisioning

For the majority of IoT services, a device that attempts to connect to a service must be authenticated by the service. Likewise, a device will often authenticate the cloud service as well to ensure secure communication. This is known as mutual authentication, and mTLS is the primary method used in IoT deployments to accomplish this. With mTLS, both the device and the cloud service will present their own TLS certificates for verification using public and private keys. Furthermore, cloud services usually require devices to be uniquely identified from their certificate. With IoT device fleets deployed to the field, how do certificates get installed onto the device in the first place? That is where device provisioning becomes an important consideration. Device provisioning involves injecting unique certificates and/or encrypted keys on a device before it is ever handled by an untrusted party. This involves using a secure facility and provisioning process that ensures devices cannot be cloned or spoofed. Often, a certificate will reside in secure storage on a device with a secure element or secure enclave to avoid tampering or replication. Devices that implement these techniques prior to deployment are far more insulated from security attacks and vulnerabilities than those that do not, so these techniques are considered an important best practice.

Device Monitoring

When an IoT device is deployed, physical access to the device is often not possible. However, there is usually a vested interest in the health and general status of each device in the fleet. After the device connects to a cloud service, owners or operators of the device will expect the device to run trouble-free. Device monitoring can help achieve smooth operations. IoT devices can be programmed to send telemetry data about their individual state to an IoT cloud service for monitoring purposes. Some examples of useful data fields include: battery drain rates, restart events, unexpected crashes, connectivity issues, temperature fluctuations, physical impact, or when certain blocks of code execute. Depending on the IoT platform, there may be built-in device monitoring services available, or this may be provided from an additional vendor used for the IoT deployment. Because the data captured could be from a numerous fleet of devices too large to visualize, queries or specific event triggers can be leveraged to indicate when there are device issues at scale. Monitoring data captured from the device can be used to improve new revisions of the device firmware or future versions of the device hardware itself.

Device Updates

One way that IoT devices can benefit from connecting to the cloud is by receiving updates to the running embedded application or firmware to improve their operation. Such IoT device updates are usually referred to as OTA (Over-The-Air) or FUOTA (Firmware Update Over-The-Air) updates. Often, this is a bespoke process depending on the device hardware capabilities and the approach chosen by the manufacturer. First, a new firmware image is produced that is tested and versioned to be deployed to the fleet of devices. Many IoT platforms will support OTA updates as part of their device or fleet management capabilities, which allows an administrator to upload new firmware images to the IoT platform for distribution. It is imperative that the updating process does not “brick” a device, which would render it unable to connect to the cloud or receive any further updates. The IoT platform will signal to the fleet that an update is available. Depending on how the device is used, the device may begin downloading the new image, or it may have to wait until it is in the proper operational state or the user agrees to the update. To improve the network efficiency of distributing new firmware images, some device update processes will only send deltas or the difference in bytes between the old and new firmware image. Once the device has retrieved the new image or delta, it will usually require a restart to point the bootloader to the new firmware. For increased security, firmware images may be digitally signed, which then requires the device to verify its authenticity. Finally, once the new firmware is in place, if an issue is detected, a rollback may be required. This is when the device goes back to running a previous version of the firmware because of a problem encountered with the new firmware. A notable point is that version differences across the fleet may introduce challenges. Can devices still at v1.0 of the firmware skip v1.1 and update to v1.2 if it has become available? What if a user chooses not to update? Some degree of pre-planning and future-proofing is needed to properly support a device update strategy.

Analytics and Insights

Beyond the telemetry data sent to the cloud for IoT device monitoring mentioned earlier, capturing data from devices that leads to business insights is one of the main reasons why businesses invest in IoT deployments. After all, data storage by itself is simply an added cost to operate and maintain with no real benefit. To turn IoT data into an asset, analytics must be applied to provide insights. For example, a smart garage door may capture its open and close state and send that data to the cloud. However, if the business wanted to know how often customers open and close their garage doors daily across the fleet, tools are needed to analyze that raw data. IoT analytics is broad terminology that may include database queries, computational analysis, statistics, or even machine learning to quantify patterns in IoT device data.

Furthermore, some type of representation of this data through reports or visualization like dashboards can provide insights into an analysis. Data scientists may be involved in developing methods to extract meaningful insights from datasets. Using the smart garage door example, a discovery might be made after the analysis that customers open and close their garage doors far more frequently than the manufacturer projected, causing shorter lifetimes of the motor and increasing warranty claims. Such insights may be valuable to the business to better understand the exact behavior and usage of an IoT device by their customers, which in turn can be utilized to provide better products and services. There are many technology options and considerations for providing analytics and insights in an IoT deployment. Some services specialize in this domain specifically, and the choices for adoption depend primarily on what is appropriate for the business needs.

Data Storage

How data is stored is an important aspect of an IoT deployment. Three main considerations for data storage are: 1) what data should be stored, 2) how will the data be accessed, and 3) what are the costs associated with storing the data. When unconnected devices become IoT devices, data that was limited to the device itself can now be sent to the cloud. The impulse is to send all of the available data from any of the sensors on the device to the cloud so that analytics can be applied. However, decisions should be made on what data should be sent while considering privacy and efficiency. The data transfer and storage costs incurred may be more than is actually required. Local processing is an important factor in reducing transfer and storage costs. If, for example, an embedded application takes readings from a temperature sensor every second, but the temperature value itself did not change for an hour, then it may not be efficient to send 3600 readings to the cloud with the same value during that time frame. Instead, perhaps the data should be sent only when the temperature value changes.

Once it is determined what data needs to be sent, there are many ways to store the data in the cloud. Data can be stored unstructured in flat files or structured in a database. Choosing the storage mechanism depends on the type of data and how the data will be accessed. If the data will be queried frequently to show real-time statistics, then a time-series database may be the right choice. However, if the data is large, not frequently accessed, and only needed for reporting or auditing, maybe a data lake would be a better option to keep the costs low.

Finally, the retention policy of the data is a key consideration. Does the data need to be stored indefinitely or can the data be retired after some period of days or months once it has served its purpose? This may depend on the type of data as well. Data that represents the control plane of the device fleet may have a different retention policy than that of the data plane which represents the device usage or sensor peripherals. There may also be local privacy and data governance considerations that affect these storage requirements.

Building Blocks

The choice of an IoT platform for an IoT deployment depends on the needs of the business. An IoT deployment may even include multiple services from different vendor platforms, using them as building blocks to architect a complete solution. Some specialized IoT platforms include capabilities that may abstract certain technical details outlined above in order to make managing fleets of devices easier. It is also worth noting that some IoT platforms may run on top of other IoT platforms, utilizing their services as building blocks to provide higher order capabilities for specific use cases. For example, an IoT platform that provides premade dashboards for visualization may run on top of another IoT platform that implements an MQTT broker that is resilient and massively scalable for device fleets. Therefore, IoT deployments and the IoT platforms themselves may be constructed using various building blocks to help make the deployment, management, and analysis of fleets more efficient in terms of resourcing, costs, and capabilities.

Tiny ML and IoT

It is a good idea to compare and contrast Tiny ML and IoT technologies in order to obviate symbiotic relationship between the two. For example, Tiny ML benefits tremendously from the advances in IoT deployment infrastructure whereas IoT benefits from Tiny ML's ability of on-device execution in terms of reducing or eliminating cloud and connectivity dependence making it attractive to verticals that require always-on execution regardless of connectivity. Tiny ML is popular or expected to gain popularity where IoT devices fall short of the requirement of connectivity, electricity, and stealth mode execution:

1. No internet connection required for operations
2. Low power consumption
3. Improved Latency
4. Battery-powered standalone solutions
5. On device execution makes it independent of cloud
6. Applications for remote areas with no connectivity or electricity.
7. Stealth mode operation
8. Enhanced privacy
9. No cyber security required for stand-alone on-device operation.
10. Reduction of elimination of bandwidth

In summary, Tiny ML and IoT are very similar when it comes to deployment and remarkably different for MLOps.

Tiny MLOps

The picture below (Figure 1) outlines Tiny MLOps to develop end-to-end Tiny ML solutions. Steps 2, 7 and 8 stand to benefit from IoT infrastructure and its secure connectivity advances.

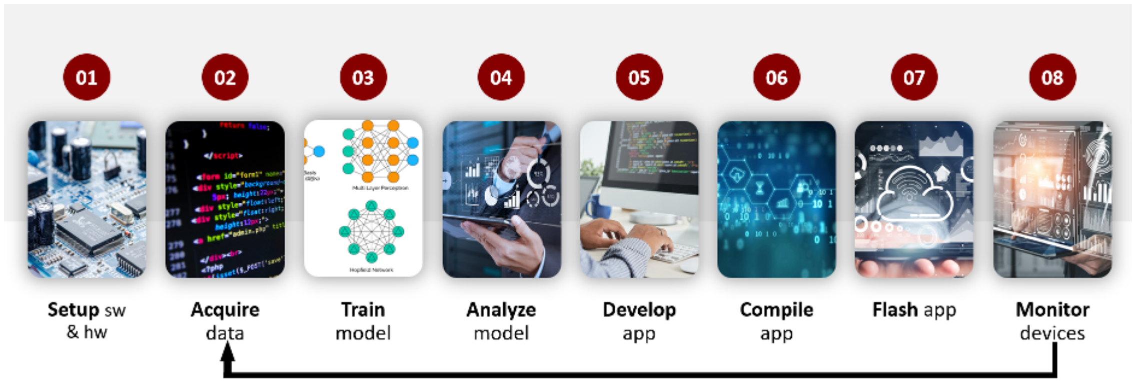

Figure 1 - Tiny MLOps - courtesy of www.ai-tech.systems

Device Connectivity

It is noteworthy to mention that on-demand device connectivity of IoT is the main reason behind its popularity in the last decade. Requirement of connectivity is what makes it less suitable for remote and connectivity degraded industries like farming, mining, shipping, space, aviation, defense and others. On-device execution of Tiny ML technology makes it not only appealing after deployment, it makes it the only choice for a select few like defense where stealth mode operation of device is of paramount importance.

Using IoT for Tiny ML Deployment

Tiny ML deployment refers to an entire MLOps deployment pipeline. There are multiple factors that are unique to a Tiny ML deployment.

1. MLDevOps pipelines on the cloud
2. MLDevOps pipeline on the device
3. ML model security, privacy and data governance

ML on the edge typically involves inference on sensor representations of the physical world. This is very different from other ML applications such as spam detection which involve a fixed input type (unicode). The physical world is constantly changing and evolving. This results in concept and bias shifts[1] which can degrade inference results. To overcome these issues, ML models on tiny devices are typically part of a MLDevOps pipeline as shown in Figure 1 above.

The cloud portion of the pipeline is shown in the figure. The pipeline starts with a new device joining the fleet and advertising its capabilities. AutoML[2] is used to produce a model that optimally uses the capabilities of the target device. In the Tiny ML space, fleets can be extremely heterogeneous and use multiple variants of the same basic model (e.g. devices may offload a portion of the compute to a local accelerator). The model is then deployed to the target device using either OTA or through a connected gateway.

IoT deployment infrastructure can be used to get inference results and raw sensor output (limited by energy and communication constraints) back to the cloud or server. These results are monitored for input statistics and

[1] Understanding Dataset Shift. How to make sure your models are not… | by Matthew Stewart | Towards Data Science

[2] Wikipedia, **Automated machine learning** (**AutoML**) is the process of automating the tasks of applying machine learning to real-world problems. AutoML potentially includes every stage from beginning with a raw dataset to building a machine learning model ready for deployment.

performance metrics (e.g. accuracy) to detect data and concept drift. Detection of drift or drop in performance would trigger a sequence of troubleshooting steps and/or retraining the model and redeployment to the device. Sometimes, this detection can be done on the device itself if the application statistics are well understood. For example, in an office building, there should be a drop in the number of people after office hours. Data from a device showing a lot of people after hours would be an indication of improper operation.

Model versioning needs to be enforced for robust Tiny ML deployments. A robust ML devops deployment would track all of the following: training code version (eg Tensorflow 2.11), Training and validation datasets, training metrics, Tiny ML model converter tool version, converted model and binary that executes converted model. A JSON or YAML version file would be preferred to store and access all these values.

Unlike an IoT application which is typically a single function, Tiny ML applications on the device can be envisaged as a pipeline composed of multiple stages such as data ingestion from sensor, data preprocessing, Tiny ML model inference, post processing, business intelligence and other monitoring.

While this pipeline could be deployed as a single binary, it is preferred to have multiple artifacts that can be individually updated for communication bandwidth and power optimization. For example, when TensorFlow Micro is used as a runtime, the model array can be separately updated without updating the rest of the code (assuming the activation sizes and other constraints can still be met).

There are two aspects to security associated with ML models. ML models are valuable intellectual property (IP) since they are typically trained with proprietary (non-open source) user data. There is thus a lot of value in the model. Preventing a hacker from stealing models from a physically accessible device is a tough problem. The second problem relates to a hacker being able to alter model behavior without detection. Since responses from sensors guide health and safety responses and decisions, this represents a serious problem. IOT deployments typically don't suffer from either of these security problems.

Privacy and data governance/sovereignty represent other challenges to ML training and deployments. Storing data securely (particularly raw sensor data) and transmitting it while meeting regulations can be particularly challenging. In addition, new government regulations on privacy and data governance require tracking the data and its usage in the cloud. Responsible AI also mandates that data from a device (used for model retraining) be tracked, versioned and associated with models that use it. In parallel, federated and on-device learning is being developed which may keep private data secured on the individual devices while reducing the bandwidth and power required to connect to the cloud.

Conclusion

This white paper explored the challenges and solutions for deploying Tiny ML applications by building on the fundamentals of IoT Deployment and MLOps. As outlined in the sections above, there are many areas that developers can use existing IoT solutions due to the commonality. However, there are many other areas where the Tiny ML aspects of the end application will require a substantially different approach than the existing IoT deployment solutions. Therefore, developers need to understand the key similarities and differences between IoT and Tiny ML devices, applications, and environments.

Successful Tiny ML applications will require developers to consciously select the necessary elements to purchase or develop for each layer of a "full stack" implementation to properly deploy their product(s). Relying on "default" solutions may work for development test cases but may hinder large scale deployment "in the field".

Appendix - Case Study

This case study provided by Stream Analyze illustrates how they handle deployment within their product platform. It is illustrative of how one company addressed the challenges described in this white paper.

Stream Analyze: enabling interactive AI model development and systematic deployment on edge devices

Stream Analyze (SA) provides the software SA Engine, capable of executing on resource constrained edge devices. SA Engine contains a main memory database and a data stream management system. In SA Engine, queries are specified using the query language OSQL (Object Stream Query Language). An OSQL *model* is a number of functions defined by queries. OSQL is declarative in that the user specifies queries that transform, filter, and combine data streams without explicitly specifying *how* to execute these queries. OSQL combines strong type checking for numerical computations with efficient search and filtering over data streams. Queries are compiled on-the-fly and immediately executed with unnoticeable delays. Using SA Engine, the user may interact with fleets of edge devices by querying these, similar to querying a database. SA Engine includes a rich model library, including mathematical, statistical, and ML models. The model library is extensible, allowing users to bring their own models.

When using SA Engine to deploy a neural network (NN) model to one or more edge devices, the NN is transferred to all the participating edge devices and stored in the SA Engine main memory database of each edge device. Thus, a deployment of a new model is essentially an update of the database of each participating edge device; no firmware update over the air (FUOTA) or other firmware changes are necessary for model deployment.

A fundamental difference between OSQL and contemporary scripting languages such as Python is that OSQL is a very high-level declarative query language where queries are optimized. By removing the need to write detailed procedural code, SA Engine allows users to focus on the *model* rather than *implementation details*. SA Engine has the freedom to optimize the declarative queries in the model. Allowing the query optimizer to automatically generate an optimized execution plan is known to be scalable and very performant. Providing a high-level specification language is also known to improve user productivity and lowers the entry bar to edge analytics and enables a larger user-base compared to solutions that use procedural scripting languages and/or FUOTA for updating code.

In Figure 2 a layer diagram of SA Engine running on a microcontroller is shown. If no operating system is present on the device, SA Engine includes necessary hardware drivers. Figure 3 shows how SA Engine runs on a regular operating system on machines such as telematics control units or cloud nodes, including containers.

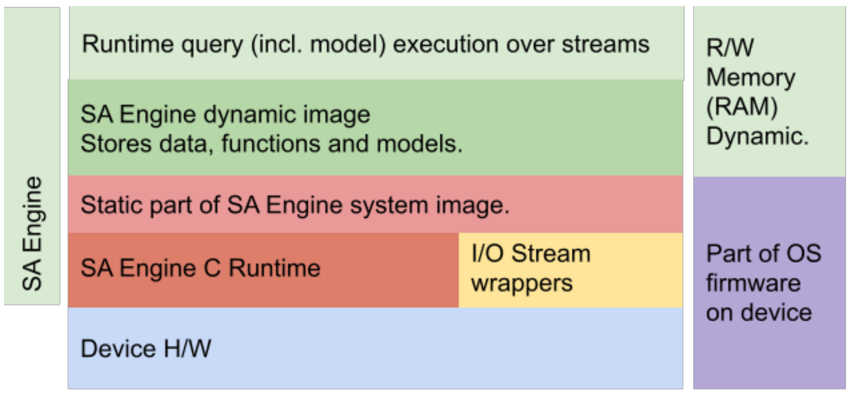
SA Engine
Runtime query (incl. model) execution over streams
SA Engine dynamic image
Stores data, functions and models.
Static part of SA Engine system image.
SA Engine C Runtime
I/O Stream wrappers
Device H/W
R/W Memory (RAM) Dynamic.
Part of OS firmware on device

Figure 2 - A layer diagram of SA Engine running on a microcontroller without operating system.

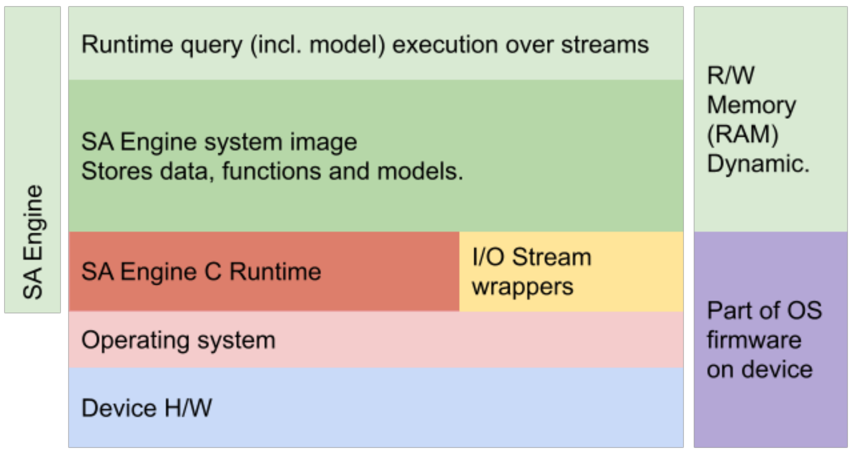
SA Engine
Runtime query (incl. model) execution over streams
SA Engine system image
Stores data, functions and models.
SA Engine C Runtime
I/O Stream wrappers
Operating system
Device H/W
R/W Memory (RAM) Dynamic.
Part of OS firmware on device

Figure 3 - Layer diagram of SA Engine running on a device with an operating system included.

SA Engine installed on edge devices connect to *SA Federation Services* instances installed in the cloud or on premise. These services are monitoring the status of all connected edge devices, and deploys queries and models on selected devices. A user interacts with SA Federation Services and its connected edge devices using *SA Studio*, a user interface with query editing support and real-time visualization facilities. SA Studio is available as a stand alone application, a browser based application, or as a plugin to Visual Studio Code. Any queries and models produced using SA Studio may be versioned using conventional code repositories.

SA Engine also integrates with continuous integration / continuous delivery (CI/CD) systems, allowing users to systematically test and deploy their models.

- Using SA Studio, the user may ask queries to one or several edge devices interactively, enabling unparalleled iteration times for the user. This solves the edge analytics problem. In addition to querying sensors, SA Engine monitors the health of the device, allowing the user insights into properties including battery drain rates and connectivity issues, and physical impact.

- By having the models inside the main memory database and running them using the dynamic runtime which allows for FUOTA-less updates of models. Users can iterate over models on the device without having to worry about bricking it. Once a candidate model is ready to be deployed, the model is tested and validated using CI/CD. In a Stream Analyze CI/CD environment, emulated or physical versions of the target edge devices are included. The combination of code repository and CI/CD environment for model management solves the model versioning problem. Then, the user may choose to deploy the model using SA Engine, or include the model in the static part of SA Engine, so that the model is deployed using regular FUOTA.

Querying fleets of edge devices

The function to send queries across vast fleets of connected industrial equipment is a very powerful tool in an edge computing and analytics context. Queries can be sent based on various parameters to instantly get real time data streams back. These data streams can immediately be visualized, analyzed, and iterated to filter out exactly the data needed for the use case at hand. A few examples are the following. See also figure x below.

A query can be sent as follows: "Send streams with vibration data for all equipment that are located in the US". Such a query is easily created with our high-level query language. In this case, two parameters are used, vibration data and location in the US.

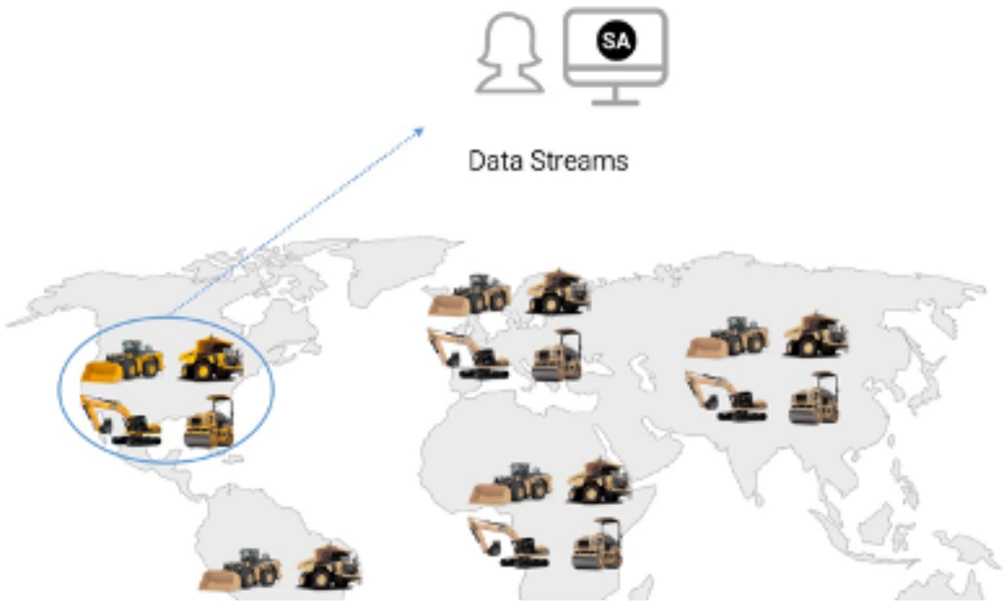

Figure 4 - Example of a query to instantly get real time data streams

It is possible to send queries combining parameters. An example is: “Send streams with vibration data for all equipment located in the US, Europe, or Australia from wheel loaders or dumpers that have an outside temperature below –20 °C”. In this case, four key parameters are used to further filter out data: vibration streams, location, type of equipment, and temperature. Queries created in this manner can freely combine and iterate different parameters. In a matter of minutes, a user can drill down into huge volumes of data across hundreds of thousands of devices and thereby filter out exactly the data needed when the data is needed.

Deployment and orchestration

The process to test and deploy models to edge devices is very similar to the process of sending queries. The process is instant where the model is sent to the engine running in the devices, bypassing the need to do full firmware updates. Only the model is sent to the devices in the deployment process whereby the firmware update process and the model update process are separated.

Testing, deployment, and orchestration can also be done based on parameters to decide to which devices a deployment should be done. When orchestrating deployments, users have access to the full distributed system of SA Engines out in the wild. This allows them to run queries over central meta-data for selecting edges to deploy to, however, it also allows them to query populations of edges of their current state, locally, and decide whether or not to deploy a model to these edges. This ensures only the devices that will run the models, download them. Saving both bandwidth and storage.

Large enterprises are likely to have a portfolio of models for different use cases and types of equipment, where deployment usually is done only to a selection of all the devices in the fleet. Orchestration is the process where deployment is done in a structured and systematic manner keeping track of which model versions are deployed to which devices.

In this example a model is deployed based on one parameter: “Deploy a vibration anomaly detection model for all equipment located in the US”. The model is deployed instantly to all equipment located in the US. Results from the deployed models in the form of streams are sent back. The results can be easily visualized in real time. A model can be iterated and redeployed within minutes if changes are needed. A model can also be configured to work autonomously on the edge devices in the case of intermittent connectivity.

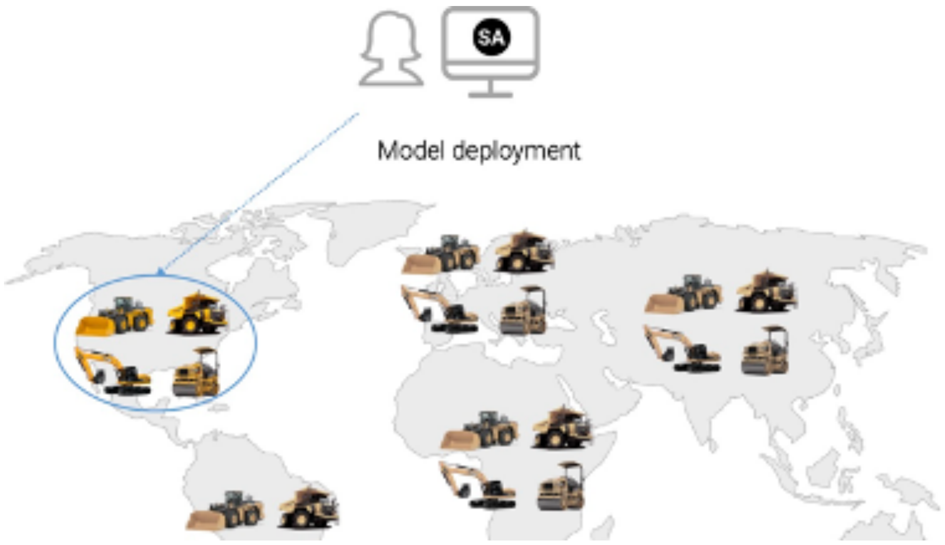

Figure 5 - Deployment and orchestration

A deployment can also be done based on several parameters where it becomes more of an orchestration process. An example is: “Deploy a vibration anomaly detection model to equipment located in the US, Europe or Australia, that are wheel loaders or dumpers and with a threshold value for vibration over a certain level”. In this case, three parameters are used: location, equipment type, and a sensor threshold value. Orchestration can be done by freely combining all available parameters; either by querying meta-data, cached information centrally or by querying the devices directly. This is a very flexible and powerful solution to manage portfolios of models across vast fleets of industrial equipment.

Authors

Elad Baram - Synaptics
Ira Feldman - tinyML Foundation
Dan Gross - Amazon Web Services
Vitaly Kleban - Everynet
Gopal Raghavan - Microsoft
Johan Risch - Stream Analyze
Rohit Sharma - AI Technology & Systems
Tomas Uppgard - Stream Analyze
Erik Zeitler - Stream Analyze

For more information, please contact Rosina Haberl rosina@tinyml.org

Other References

Design principles for unified edge device architecture - Microsoft Community Hub

"Essentials of Edge Computing", NXP