Architectural Considerations and Selected Technologies for Machine Learning at the Edge

Joseph Fuerst, Dhruv Bohra,
Elisabeth Nguyen and Ann Chervenak

The Aerospace Corporation

February 28, 2023

Motivation: Support Machine Learning at the Edge

- **Background: Edge Computing**
  - *A distributed computing paradigm that utilizes both Cloud data centers and devices at the edge of the network (e.g., systems, sensors, routers, and satellites)*
  - *Edge computing can improve the performance of analytics by utilizing the resources of edge devices, including storage, networking and computation, to:*
    - Perform data collection, analysis and filtering on the edge device, without requiring data to be transferred to the Cloud data center
    - Eliminate or reduce latency and network traffic between edge devices and the cloud data center, since less data is sent from the edge to the cloud
    - Enable faster local decision-making at the edge
    - Support intermittent connectivity or disconnected operations between edge devices and cloud data centers
- **Our focus: Enabling Machine Learning Applications to run at the edge**
  - Explore architectural considerations for deploying ML models to space-based and ground-based edge devices
  - Explore training and inference tradeoffs for edge devices

Why Machine Learning (ML) at the Edge?

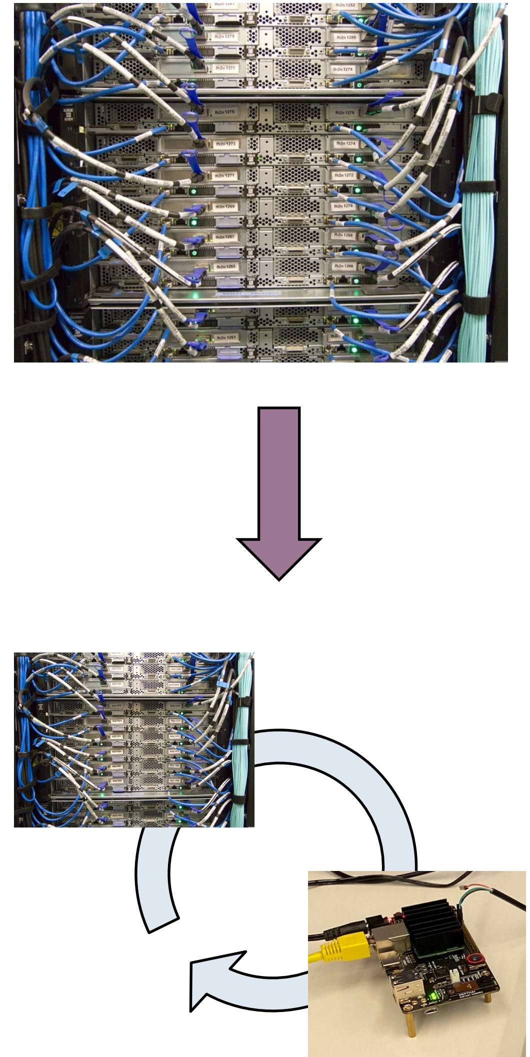
- Data preprocessing and filtering
  - *Need less Size, Weight and Power (SWaP) for storage and downlink*

- Onboard tip and cue
  - *Coordinate different sensors for edge data fusion*
- Faster reaction times
  - *Support autonomous control*

- Resilience
  - *Less reliance on ground input*

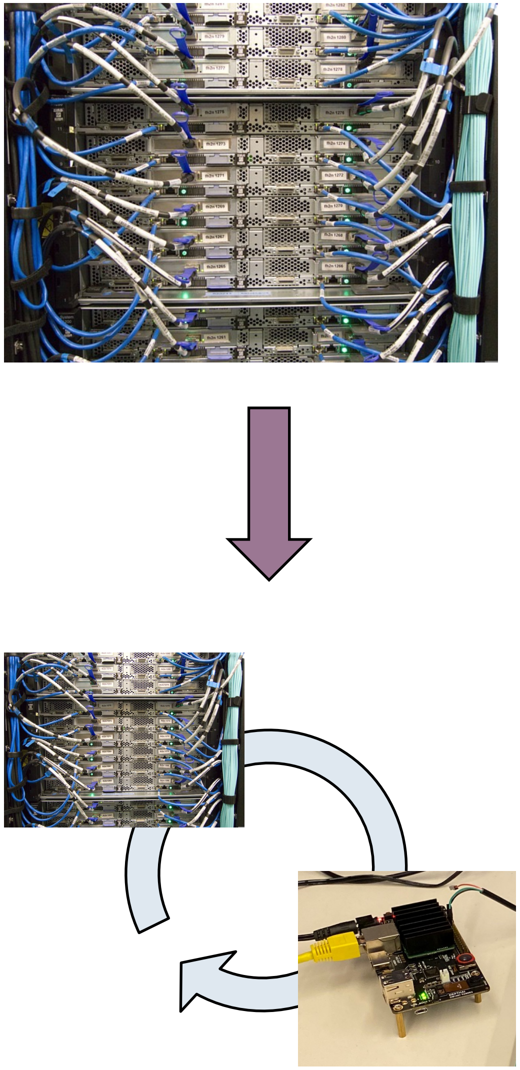
Edge ML Challenges

- Traditional training is not feasible on edge devices
  - *Don't have storage capacity for big data*
- Inference often needs to be optimized
  - *Shallower networks, lower-precision models*
- Edge devices may need to be deployed on specific frameworks
  - *Model needs to be ported to available framework*
- MLOps pipeline may need to include ground
  - *Not just training, but validation and retraining, may require storage and processing capacity only available on the ground*



Edge ML Challenges

- Traditional training is not feasible on edge devices
  - *Don't have storage capacity for big data*
- Inference often needs to be optimized
  - *Shallower networks, lower-precision models*
- Edge devices may need to be deployed on specific frameworks
  - *Model needs to be ported to available framework*
- MLOps pipeline may need to include ground
  - *Not just training, but validation and retraining, may require storage and processing capacity only available on the ground*



Offline vs. Online Learning

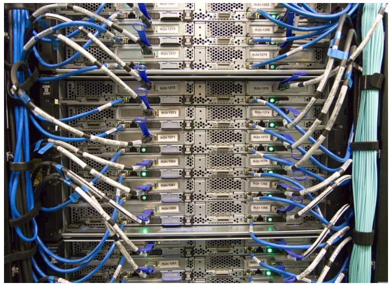
Well-defined situations:
Offline Learning

- Training done infrequently, in batches
- Humans label training data offline
- Processing capacity and storage dominate; training and retraining are likely centralized

Dynamic/uncertain situations:
Online Learning

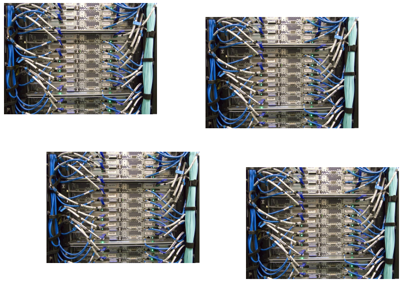
- Training done continuously at the edge
- May not be feasible for humans to label all training data
- May need multiple cooperating sensors to capture full operational context
- Security and/or privacy concerns may limit where data can be distributed
- Bandwidth to move data at the edge dominates

Centralized, Distributed, and Federated Learning



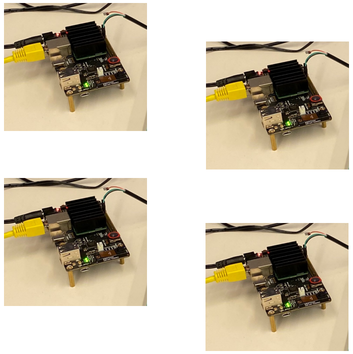
Centralized Learning:

- Data is brought to a central location
- Training is done at that location



Distributed Learning:

- Training is done at multiple locations
- Each location has a predefined subset of all training data



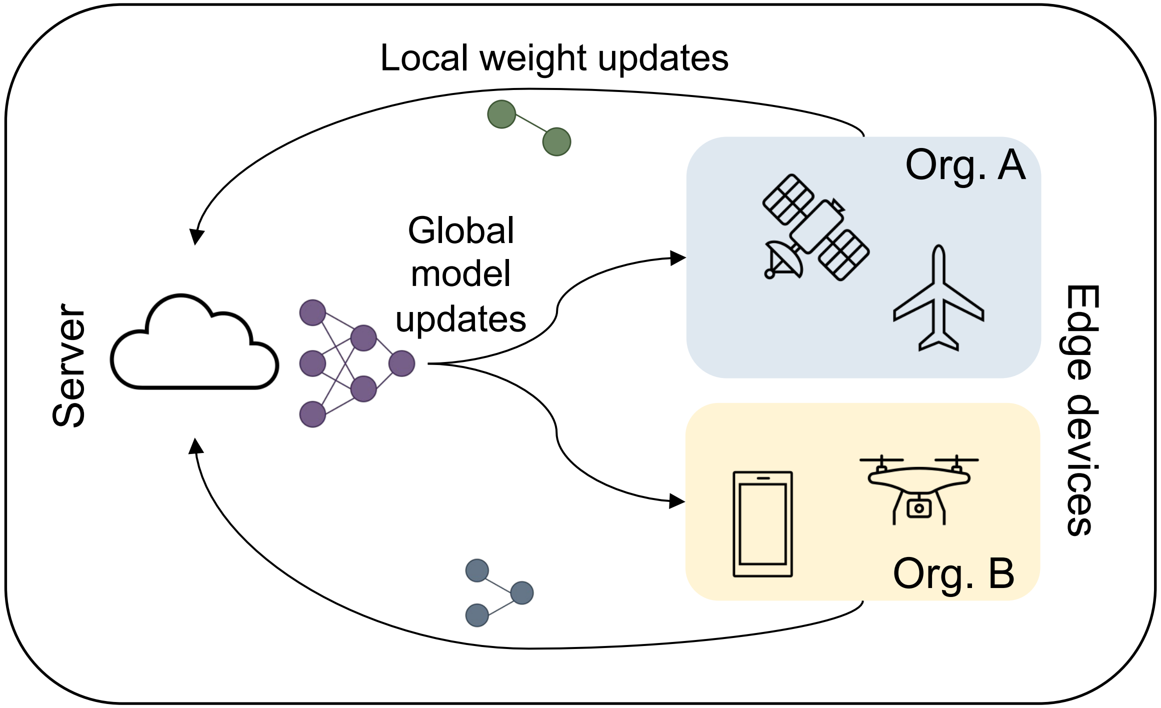
Federated Learning:

- Training is done at/near the edge

Federated Learning

Edge-Distributed, Private Learning on Heterogeneous Devices

- Server calculates and distributes a global model
- Edge sensors calculate individual sets of model updates
  - *Trains model in multiple iterations at different sites*
  - *Removes need to pool data into a single location*
  - *Sensor subsets may be aggregated at the edge*
  - *Possible to implement deeper privacy preserving techniques*
- Edge sensors send model updates to server to recompute global model
  - *Model stays roughly synchronized*
- Primarily for use with unsupervised and semi-supervised learning
  - *Process for labeling data does not work well with federated approach*

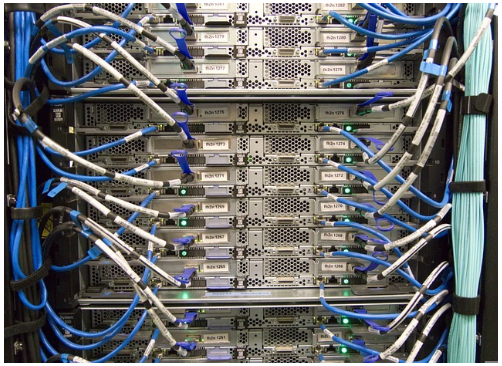


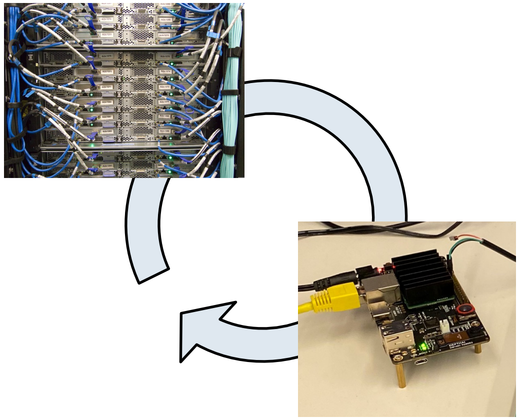
Federated Online Learning for Responsive Edge ML

- Federated learning advantages
  - *Low network bandwidth needs*
  - *Maintains data privacy*
  - *Enables devices to participate in training intermittently, when conditions permit*
- Online learning advantages
  - *Much more responsive to environmental context (does not require collecting and batching new data for retraining)*
  - *Training occurs on data streams – does not require large amount of storage*
- Disadvantages
  - *Only some ML problems can be solved by online learning*
  - *Models may be less accurate*
  - *More vulnerable to data skew and/or bad actors*
  - *Federated learning causes slower model convergence, and models may fail to converge altogether*

Edge ML Challenges

- Traditional training is not feasible on edge devices
  - *Don’t have storage capacity for big data*
- Inference often needs to be optimized
  - *Shallower networks, lower-precision models*
- Edge devices may need to be deployed on specific frameworks
  - *Model needs to be ported to available framework*
- MLOps pipeline may need to include ground
  - *Not just training, but validation and retraining, may require storage and processing capacity only available on the ground*

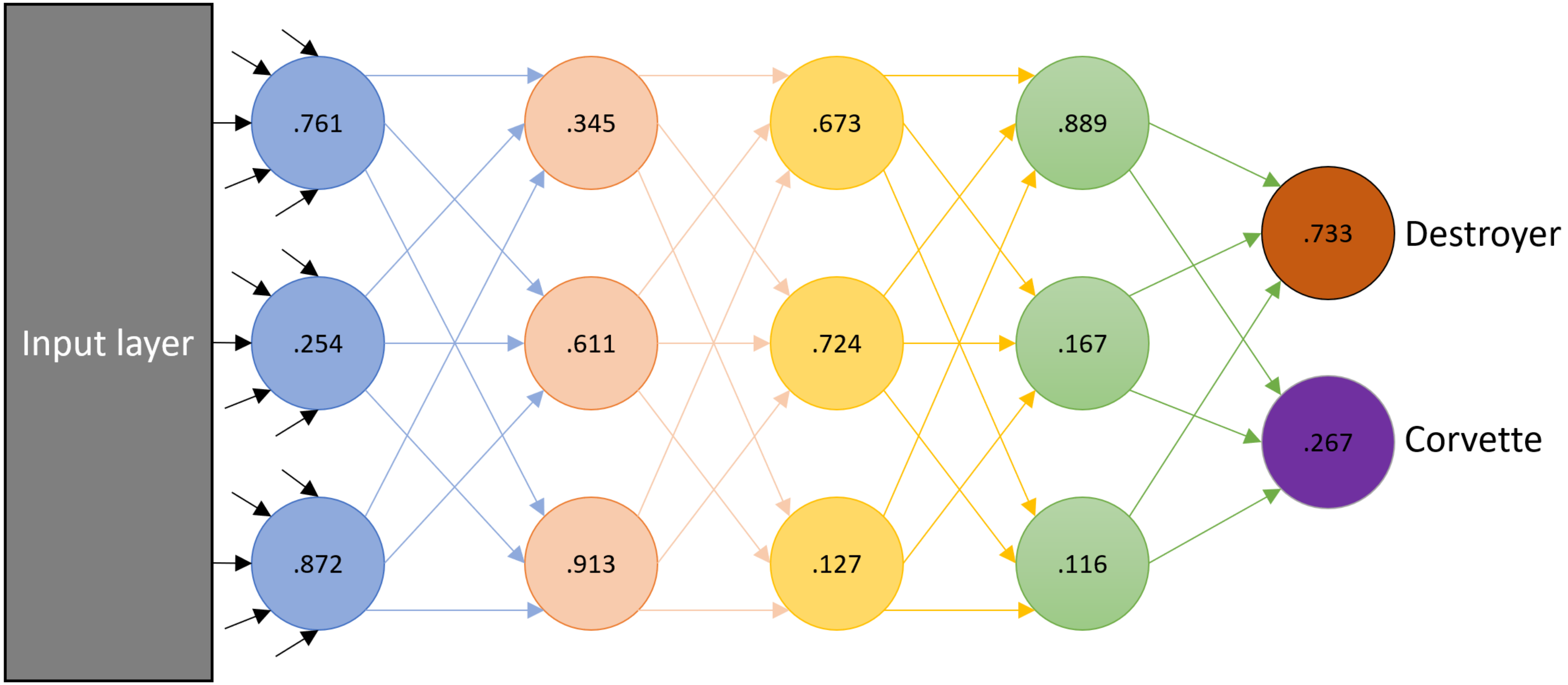




Trained Networks Are Initially Inefficient

All trained neural networks can be optimized

- It is possible to accomplish the following, while closely maintaining baseline accuracy
  - Decrease inference latency, compressed size, and memory usage
  - Increase inference throughput and accelerator compatibility

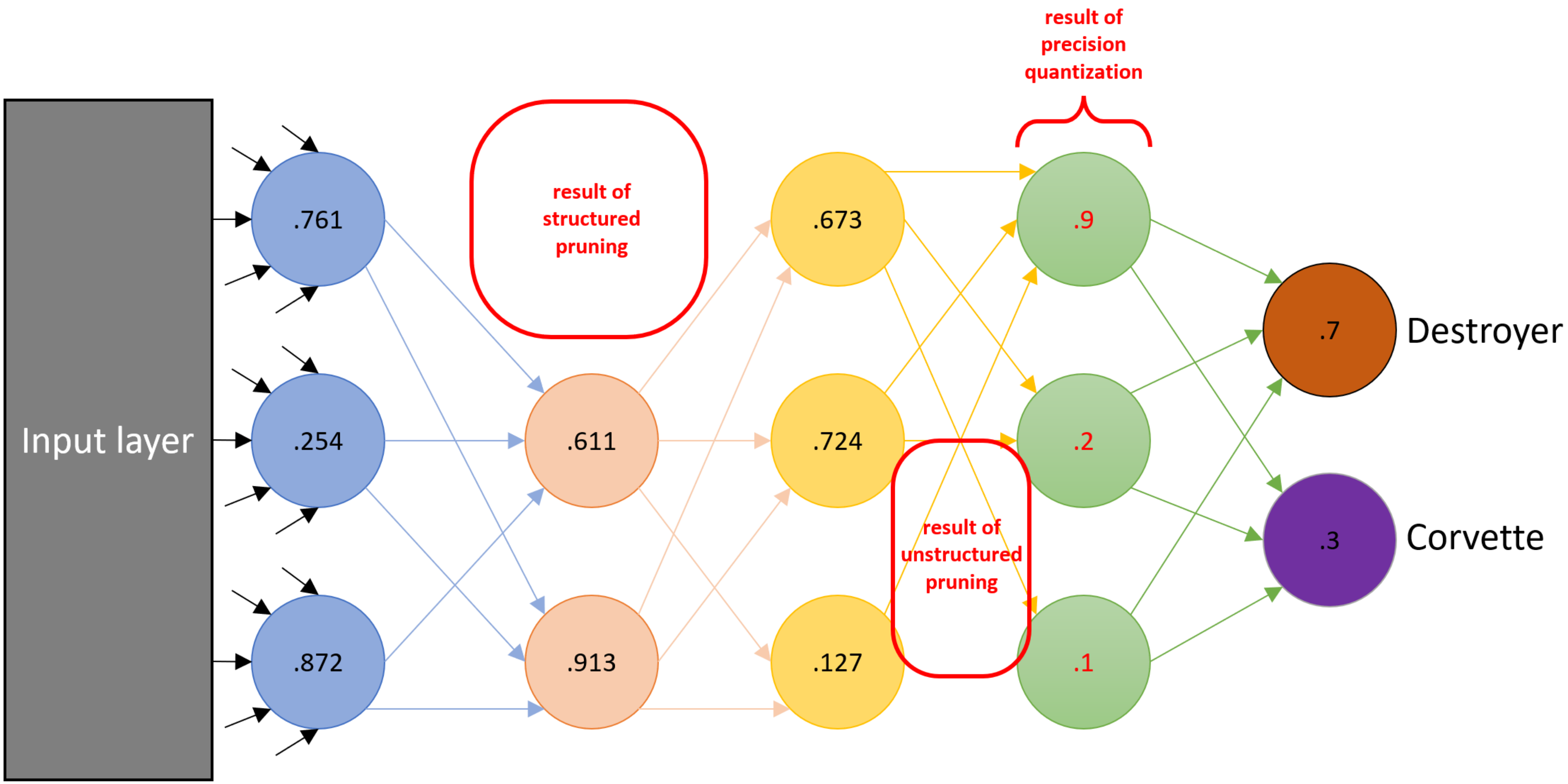


Recompiling Models for SWaP-Constrained Edge HW

- TensorFlow Lite (TFL): toolchain for destructively optimizing edge models
  - *Shrinks model size and computational demand at the cost of accuracy*
  - *Packaged with a TensorFlow pipe, allowing for high compatibility with models natively trained in TensorFlow*
  - *Widely used; supported by many different AI accelerators*
- Destructive techniques used:
  - *Weight pruning – setting some model weights to zero*
  - *Weight clustering – replacing a cluster of weights with a single centroid weight*
  - *Precision quantization – rounding or removal of decimals*
  - *Range quantization – down-converting the bit-count of weights (e.g., 32-bit to 8-bit)*
- Other similar toolchains exist, e.g., PyTorch
- Often re-optimization is a prerequisite to using an accelerator
  - E.g., Vitis AI requires Vitis 8-bit quantization
  - E.g., Google's Coral TPU-based products assume TFL optimization

Recompiling Models for SWaP-Constrained Edge HW

Neural Network optimized by TFL's pruning and quantization techniques



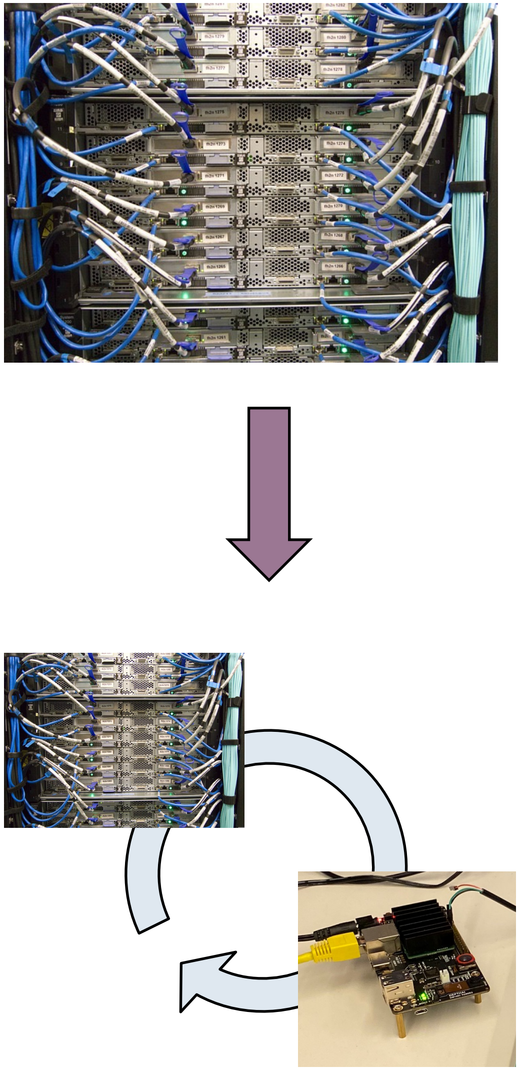
Optimizing Model Transmission

Non-destructive optimization techniques for large models

- Above techniques are destructive
  - *Require model changes; may result in loss of accuracy*
- If a drop in model accuracy is unacceptable, alternative methods include:
  - *Break model up into weights, sends only weights over the network, build architecture on the edge device*
    - Negligible improvements vs. sending the entire model
  - *Break model into chunks of arbitrary size*
    - Enables model transmission over slower or intermittent links
    - Still requires substantial bandwidth to transmit entire model
- Non-destructive methods do not substantially save on bandwidth, and do not address size, weight and power limitations of edge processing hardware
  - *For edge models, destructive methods are likely to be needed*
  - *Particularly if edge models require frequent updates*

Edge ML Challenges

- Traditional training is not feasible on edge devices
  - *Don't have storage capacity for big data*
- Inference often needs to be optimized
  - *Shallower networks, lower-precision models*
- Edge devices may need to be deployed on specific frameworks
  - *Model needs to be ported to available framework*
- MLOps pipeline may need to include ground
  - *Not just training, but validation and retraining, may require storage and processing capacity only available on the ground*



XPU Challenges

Accelerator compatibility and heterogeneous compute

- Unlike in commercial datacenters, not all edge processing systems are x86 hosts paired with enterprise-class GPUs
- Many processor categories exist, often requiring their own runtimes, such as:

Accelerator Category	Example	Runtime
Vision Processing Unit (VPU)	Intel's MyriadX	OpenVINO
Accelerated Processing Unit (APU)	AMD Vega 10	ROCm
Graphics Processing Unit (GPU)	ARM Mali	LLVM-based
Tensor Processing Unit (TPU)	Google Coral	EdgeTPU
Deep Learning Processor Unit (DPU)	Xilinx AI accelerator FPGA IP core	Vitis AI
Neuromorphic Processing Unit (NPU)	Intel Loihi	NxSDK/Lava

- Accelerator choice(s) based on mission needs, but optimally redeveloping and deploying a model using each toolchain is increasingly difficult

Growing need for inter-platform multi-vendor model translation and runtime build tools

Recompiling models to run on specific Edge HW

Simplified Apache TVM workflow diagram

ingest from any popular framework and convert to high-level expression

Load with "Relay"
ingest trained model and translate into intermediate representation (IR)

1

pass mathematical IR to AutoTVM; uses ML to discover optimal graph

Tune with "Ansor"
non-destructively reoptimize neural network graph and schedule

2

synthesize runtime binary using code generator of choice

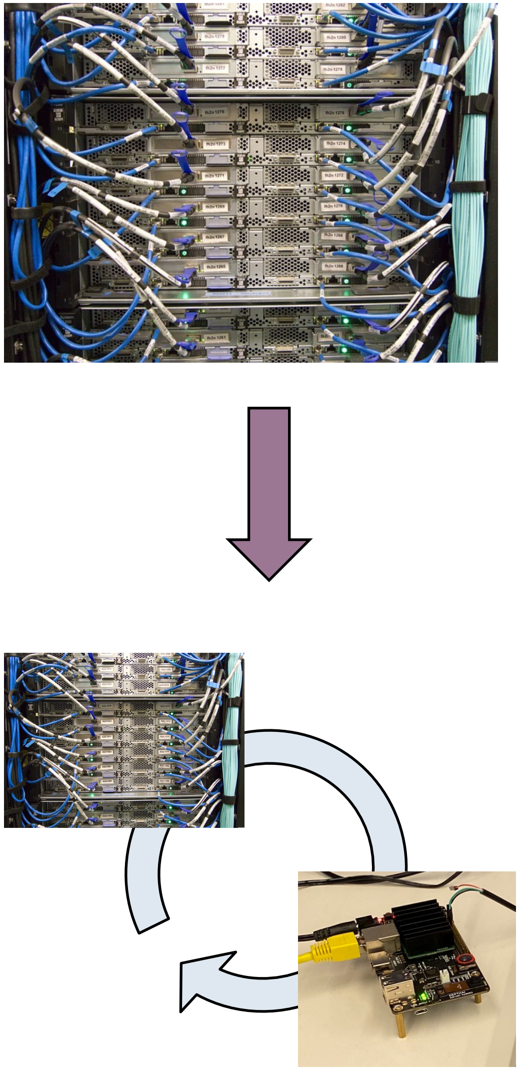
Compile with "CodeGen"
connect to a backend and generate the compiled artifact

3

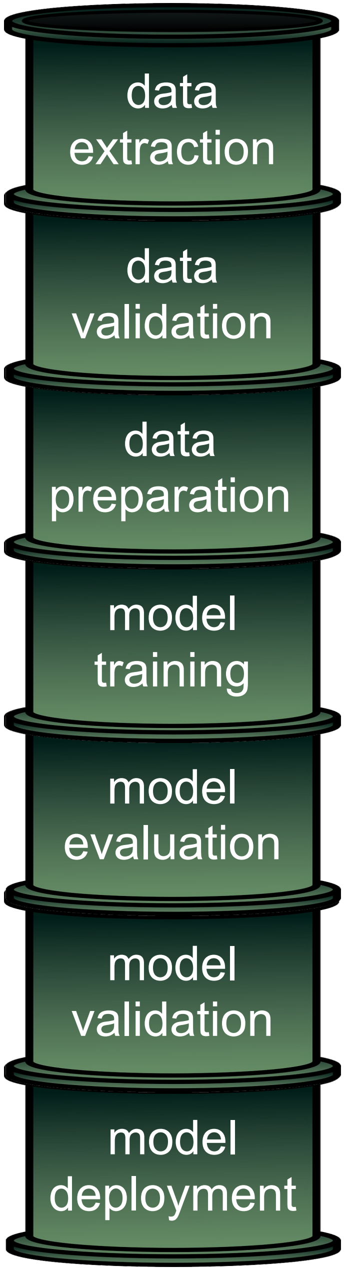
Apache TVM allows for execution on target HW

- Multiple forks and related projects are supported by academic and industry organizations
  - *μTVM: Supports baremetal C code (no operating system)*
  - *TVM Runtime: Hardware target agnostic C++ runtime for TVM-optimized models*
  - *TVM VTA: Configurable TVM-enabled FPGA deep learning accelerator and interface*

Edge ML Challenges



- Traditional training is not feasible on edge devices
  - *Don't have storage capacity for big data*
- Inference often needs to be optimized
  - *Shallower networks, lower-precision models*
- Edge devices may need to be deployed on specific frameworks
  - *Model needs to be ported to available framework*
- MLOps pipeline may need to include ground
  - *Not just training, but validation and retraining, may require storage and processing capacity only available on the ground*



MLOps Pipeline

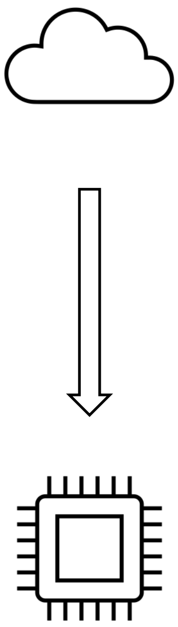
- Experience with operational ML has shown that models typically need to be updated
  - *In both the offline and online learning cases*
  - *Additional data may become available to better train the model*
  - *The operating context may change*
  - *The model may need to be deployed in new or expanded operating contexts*
- Updates are often multi-step activities, occurring in a “pipeline”
- Pipelines can be complex, and automation is helpful for both consistency and ease of use

Deploying Models

- Updates take two main forms:
  - *Model structure, e.g., the neural network itself*
  - *Model parameters, e.g., neural network node weights*
- Deployment approach depends on type of update and model server
  - *Model server could range from a simple front end to a production environment (e.g., PyTorch)*
  - *Parameter update may be achievable by sending new parameters to a running model server*
  - *Model structure update requires restarting the server*
- Updates are significantly faster and easier if the models are containerized
  - *Keeps necessary libraries, etc. together with the models to avoid version mismatches*
  - *Host operating system + container runtime are designed to easily start/stop models*
- Orchestrator (Kubernetes) can automatically deploy updated containers onto available hardware

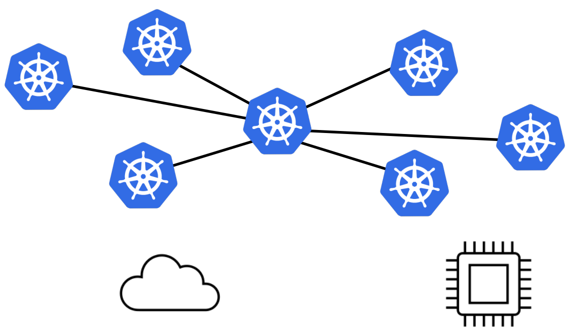
Deploying Models to the Edge

- At present, ML and orchestration are both predominantly done in the cloud
- We analyzed tools and methods to standardize model deployment to edge devices in a way that preserves portability with cloud deployments
- Focused on Kubernetes for container orchestration
  - *Use Kubernetes to move cloud-built container onto edge-based hardware*
- Assessed the following focus areas:
  - *Dealing with intermittent connection loss*
  - *Portability of cloud-native applications*
  - *Size considerations for restricted/constrained environments*
  - *Compatibility with specialized edge processors*
- Tested functionality with simple ML applications

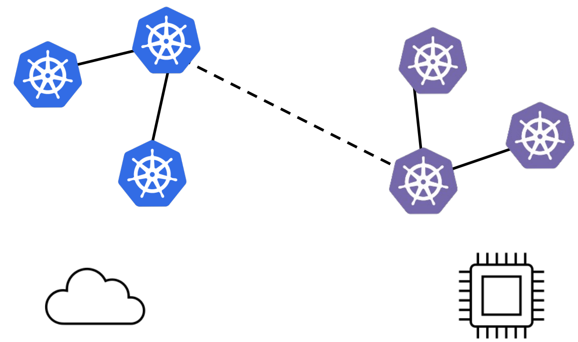


Cloud/Edge Cluster Deployment Methods

Single cluster
Edge devices included as nodes



Multi-cluster
Single cluster at each edge location



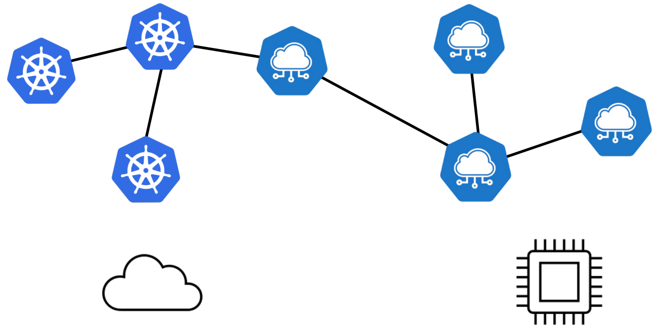
- ☑ Less overhead: cloud handles control plane and management
- ☒ More overhead: must handle downtime, syncing, etc. for intermittently connected edge nodes

- ☒ More overhead: networking, management, monitoring separate clusters
- ☒ None of the benefits to intra-cluster communication
- ☒ More overhead on edge devices to support control plane management

KubeEdge

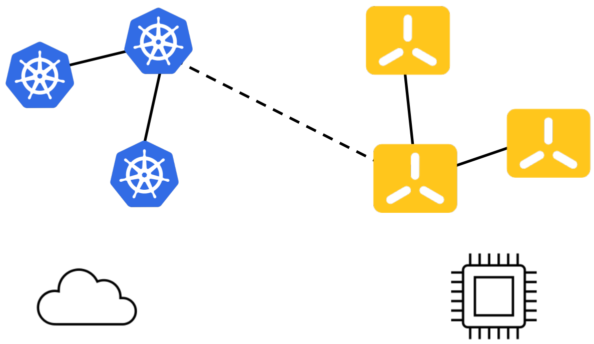
- Extends orchestration capabilities to hosts at Edge
- Enables Kubernetes native API at the edge
- Bidirectional communication and coordination between cloud and edge nodes
- Autonomous operation of edge nodes even during disconnection from cloud
- Low resource requirements, memory footprint ~70MB
- Native support of x86, ARMv7, ARMv8
- MQTT communication protocol handles IoT workloads and unreliable networks
- Findings
  - *Lack of maturity at the time we worked with it*
  - *Continued development may improve usability*
- Example use cases found here:

  https://github.com/kubeedge/examples

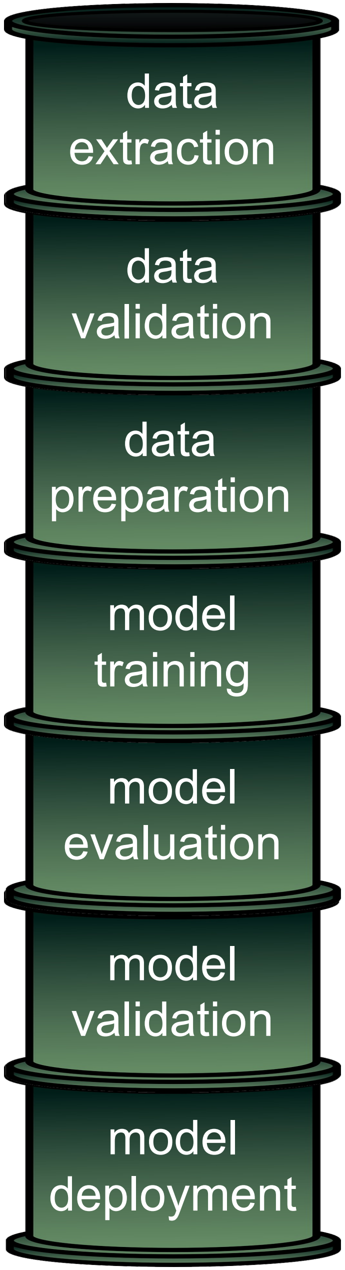


k3s

- Kubernetes variant that helps in accelerating edge computing
- Small size project (<100MB)
- Creates an edge cluster fully separate from the cloud cluster but still able to execute the same payloads
- Consists of a server and agent connected through Tunnel Proxy
  - *K3s components operate in a single process, unlike k8s.*
- Quick (<90s) spinup time for clusters
- Findings
  - *K3s is ideal for edge situations with high latency or extremely limited storage/compute/memory requirements*

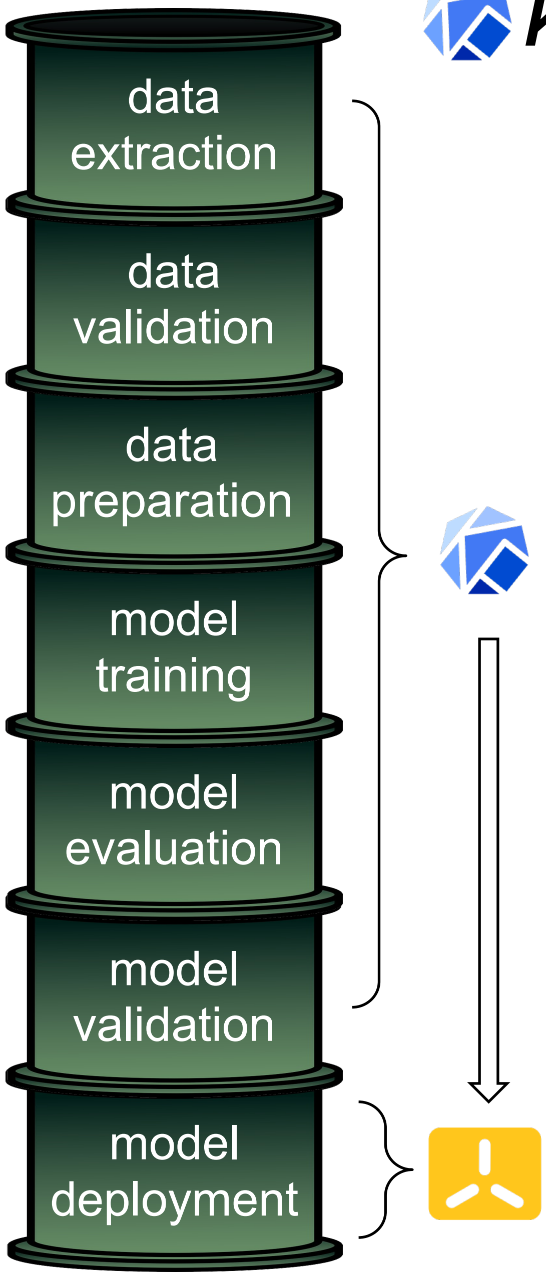


MLOps Pipeline



- Multi-stage activities such as those in the MLOps pipeline are often built into containers
  - *Enables process to be updated on the fly*
- Several tools also exist to specifically leverage orchestrators to perform MLOps
- Kubeflow: tool that uses orchestrator to manage pipeline
  - *End-to-end MLOps architecture*
  - *Becoming a widely used standard for deploying ML payloads on the cloud*
- We investigated KubeFlow for edge compatibility

Kubeflow for the Edge



- Specific concerns for the edge:
  - *Limited bandwidth may necessitate sending a compiled model or partial model rather than full updates each time*
  - *If updates are routine, MLOps solution supporting edge hardware should be identified*
- Findings:
  - *Not conscious of compute/memory/storage constraints*
  - *Not suitable for direct deployment with edge-friendly Kubernetes (k3s, KubeEdge)*
- Potential alternative:
  - *Use Kubeflow to update and validate model*
  - *Separately, use k3s or KubeEdge to deploy completed model*
  - *I.e., eject payload from the Kubeflow pipeline as a final step*

Summary of Findings for Edge ML Architectures

- Training
  - *Offline learning still likely to be done in the cloud*
  - *Online learning may be more effectively done at the edge*
  - *Federated learning takes advantage of edge device locality and preserves data privacy*
- Inference
  - *Models often need to be optimized to run on edge hardware*
  - *Various frameworks provide capabilities for this*
  - *Optimization may result in loss of precision*
- Frameworks
  - *Embedded accelerators are heterogeneous*
  - *Often require model compilation on specific framework directed to target device*
- MLOps
  - *Edge MLOps often requires a combination of cloud and edge capabilities*
  - *Edge-friendly Kubernetes variants can facilitate model deployment to edge*
  - *May need to execute most of the MLOps pipeline in the cloud and then deploy to edge as a separate step*

References: Papers/Links

- **FedAvg** paper: https://arxiv.org/abs/1602.05629
  - McMahan, Brendan, et al. "Communication-efficient learning of deep networks from decentralized data." *Artificial intelligence and statistics*. PMLR, 2017.
- **FedProx** paper: https://arxiv.org/abs/1812.06127
  - Li, Tian, et al. "Federated optimization in heterogeneous networks." *Proceedings of Machine Learning and Systems* 2 (2020): 429-450.
- **q-FedAvg** paper: https://arxiv.org/abs/1905.10497
  - Li, Tian, et al. "Fair resource allocation in federated learning." *arXiv preprint arXiv:1905.10497* (2019).
- **per-FedAvg** paper: https://proceedings.neurips.cc/paper/2020/file/24389bfe4fe2eba8bf9aa9203a44cdad-Paper.pdf
  - Fallah, Alireza, Aryan Mokhtari, and Asuman Ozdaglar. "Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach." *Advances in Neural Information Processing Systems* 33 (2020): 3557-3568.
- **Federated Multi-Task Learning**: https://arxiv.org/abs/1705.10467
  - Smith, Virginia, et al. "Federated multi-task learning." *Advances in neural information processing systems* 30 (2017).
- **Advances and Open Problems in Federated Learning**: https://arxiv.org/abs/1912.04977
  - Kairouz, Peter, et al. "Advances and open problems in federated learning." *Foundations and Trends in Machine Learning* 14.1–2 (2021): 1-210.

References: Papers/Links (cont.)

- Towards Federated Learning at Scale
  - Bonawitz, Keith, et al. "Towards federated learning at scale: System design." *Proceedings of Machine Learning and Systems* 1 (2019): 374-388.
- Improving situational awareness with collective artificial intelligence over knowledge graphs
  - Jiang, Meng. "Improving situational awareness with collective artificial intelligence over knowledge graphs." *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications II*. Vol. 11413. SPIE, 2020.
- In-Edge AI: Intelligentizing Mobile Edge Computing, Caching and Communication by Federated Learning
  - Wang, Xiaofei, et al. "In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning." *IEEE Network* 33.5 (2019): 156-165.
- Adaptive Federated Learning in Resource Constrained Edge Computing Systems
  - Wang, Shiqiang, et al. "Adaptive federated learning in resource constrained edge computing systems." *IEEE Journal on Selected Areas in Communications* 37.6 (2019): 1205-1221.
- Model poisoning attacks against distributed machine learning systems
  - Tomsett, Richard, Kevin Chan, and Supriyo Chakraborty. "Model poisoning attacks against distributed machine learning systems." *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*. Vol. 11006. SPIE, 2019.

References: Papers/Links (cont.)

- Mitchell, Nicole, et al. "Optimizing the communication-accuracy trade-off in federated learning with rate-distortion theory." (https://arxiv.org/abs/2201.02664) (2022).

List of papers by topic

- https://github.com/chaoyanghe/Awesome-Federated-Learning

Federated Learning Frameworks:

- Tensorflow Federated: https://www.tensorflow.org/federated
- PySyft by OpenMined: https://github.com/OpenMined/PySyft
- Flower: https://flower.dev/