

ECML PKDD 2021 Tutorial

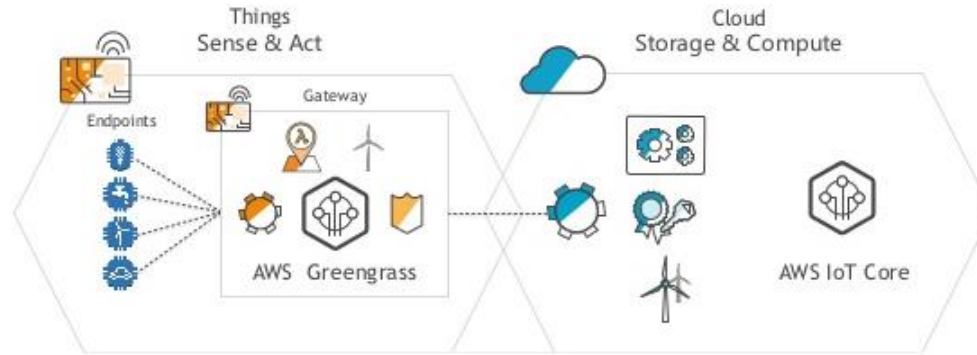
Machine Learning Meets Internet of Things: From Theory to Practice

Part II: Creating ML-based Self-learning IoT Devices

Bharath Sudharsan

A World Leading SFI Research Centre

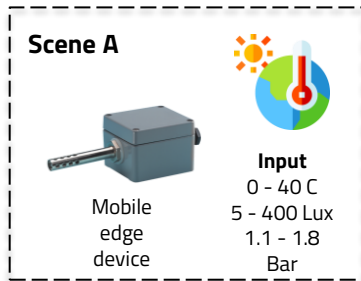




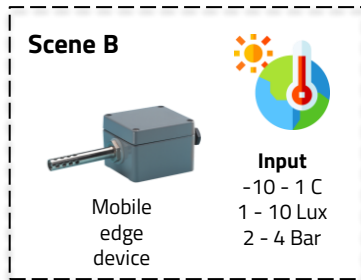
A typical IoT scenario: Edge devices (Endpoints) depends on cloud for inference and model updates

To improve inference accuracy, edge devices log unseen data in cloud. Initial model re-trained then sent as OTA update

- ✓ Edge devices hardware cost increase - wireless module addition
- ✓ Increases cyber-security risks and power consumption
- ✓ Not self-contained ubiquitous systems since they depend on the cloud services for inference and re-training
- ✓ Latency, privacy, other concerns



Accuracy: 92 %



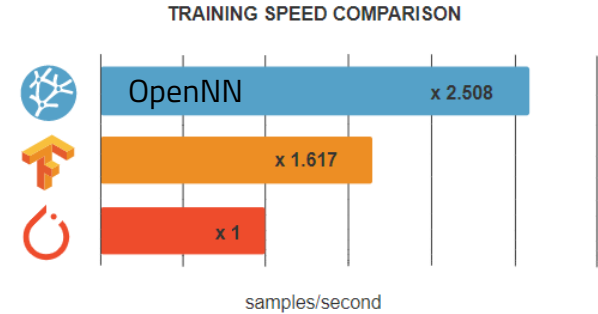
Accuracy: 54 %

IoT edge devices exposed to new scenes/environments

- In the real-world, every new scene generates a fresh, unseen data pattern
- When the model deployed in edge devices sees such fresh patterns, it will either not know how to react to that specific scenario or lead to false or less accurate results
- A model trained using data from one context will not produce the expected results when deployed in another context
- It is not feasible to train multiple models for multiple environments and contexts

On Device ML Model Training

- To enable the edge devices to learn offline after deployment - ML model deployed and running on the edge devices need to be re-trained or updated at the device level



Top ML Frameworks for running ML models on MCUs. It does not yet support training models on MCUs

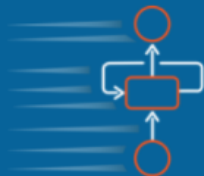
- ML frameworks like TensorFlow Micro, FeatherCNN, Open-NN, etc. do not yet enable training models directly on MCUs, small CPUs and IoT FPGAs
- Currently models are trained on GPUs then deployed on IoT edge devices after deep compression/optimization

On Device ML Model Training



Bonsai

A tree based classification/regression algorithm.



FastCells

A new class of RNN cells (FastRNN & FastGRNN) carefully designed for resource efficiency



EMI-RNN

A model independent algorithm for training smaller and faster RNN cells (GRU, LSTM, FastRNN and others).



ProtoNN

A k-nearest neighbours inspired prototype based classification/regression algorithm.

Microsoft EdgeML provides Algorithms and Tools to enable ML inference on the edge devices

- Bonsai: non-linear tree-based classifier which is designed to solve traditional ML problem with 2KB sized models
- EMI-RNN: speeding-up RNN inference up to 72x when compared to traditional implementations
- FastRNN & FastGRNN: can be used instead of LSTM and GRU. 35x smaller and faster with size less than 10KB
- SeeDot: floating-point to fixed-point quantization tool including a new language and compiler

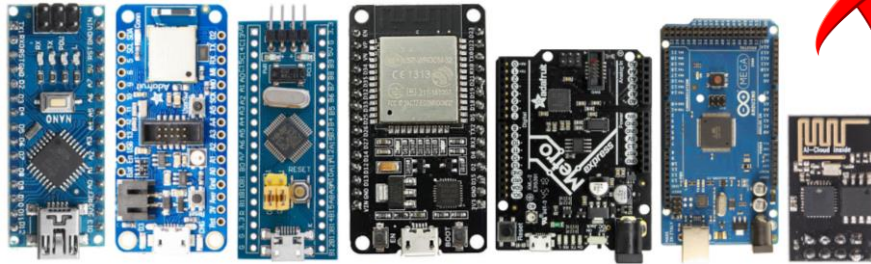
On Device ML Model Training



Powerful CPU + basic GPU based
SBCs (single board computers)



Billions of IoT devices are
designed using such MCUs and
small CPUs



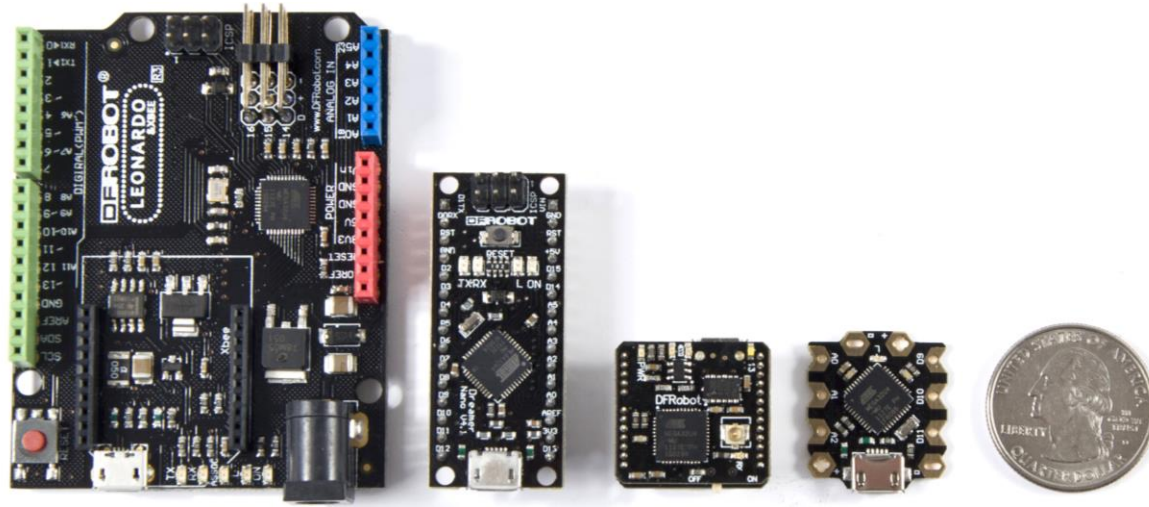
MCU1	MCU2	MCU3	MCU4	MCU5	MCU6	MCU7
ATmega328P	nRF52840	STM32f103c8	ESP32	ATSAMD21G18	ATmega2560	ESP8266
8 kB SRAM	256 kB SRAM	20 kB SRAM	520 kB SRAM	32 kB SRAM	8 kB SRAM	32 kB SRAM
32 kB Flash	1 MB Flash	128 kB Flash	4 MB Flash	256 kB Flash	256 kB Flash	1 MB Flash
@ 16 MHz	@ 64 MHz	@ 72 MHz	@ 240 MHz	@ 48 MHz	@ 16 MHz	@ 80 MHz



- Numerous papers, libraries, algorithms, tools exists to enable self-learning and re-training of ML models on better resourced devices like SBCs
 - ✓ Due to ML framework support i.e., TF Lite can run on SBCs and not on MCUs
- SEFR paper: Classification algorithm specifically designed to perform both training and testing on ultra-low power devices
- ML in Embedded Sensor Systems paper: Enable real-time data analytics and continuous training and re-training of the ML algorithm

- After real-world deployment, can the highly resource-constrained **MCU-based IoT devices** autonomously (offline) improve their intelligence by performing onboard re-training/updating (self-learning) of the local ML model using the live IoT use-case data?

ML Model Training on MCUs



Our framework algorithms enables training ML models on such highly resource-constrained, ultra-low-cost, tiny devices

Edge2Train, Train++, ML-MCU are the proposed frameworks that contain novel algorithms to enable MCU-based IoT devices to continuously improve by learning from the fresh real-world data patterns seen after deployment

Edge2Train is baseline -> **Train++** conserves memory + speedup training and inference -> **ML-MCU** is state-of-the art

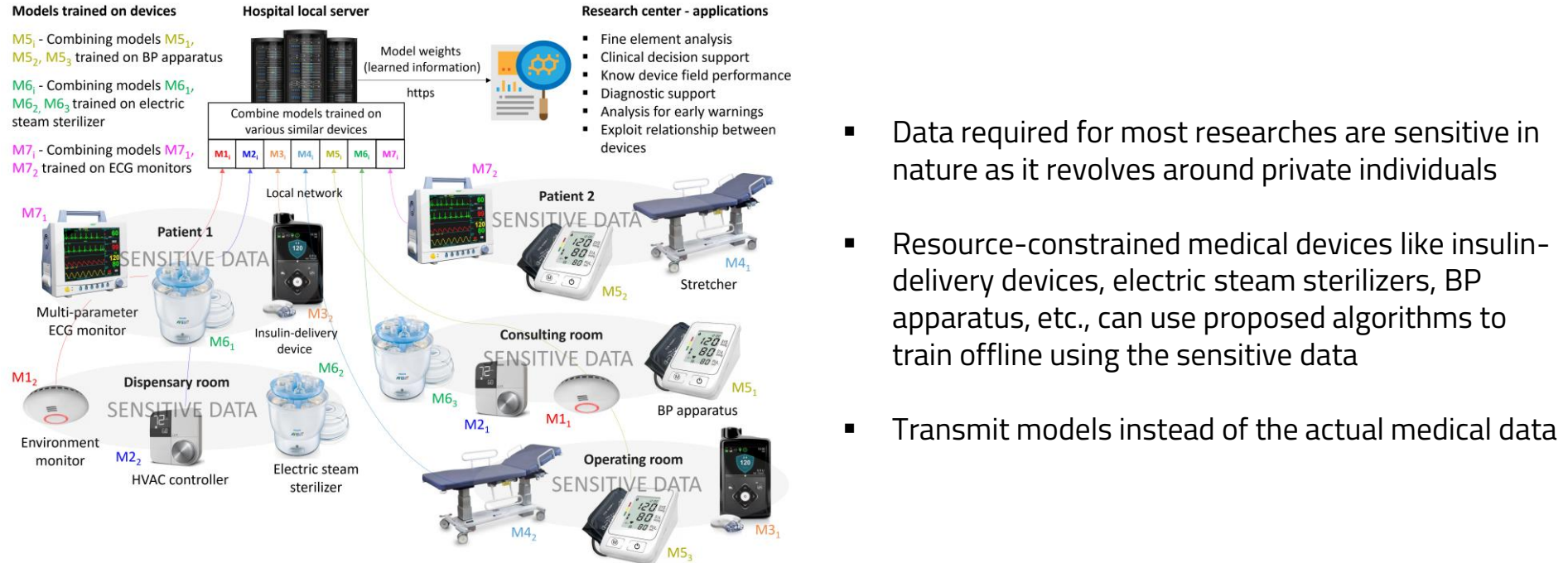
Use case 1: Self-learning HVACs



Create self-learning HVACs for superior thermal comfort

- HVAC controllers are resource-constrained IoT edge devices. For example, Google Nest, Echobee, etc.
- Every building/infrastructure has differences. Standard/one-size-fits-all strategy fails to provide superior level of thermal comfort for people
- Proposed algorithms can make HVAC controllers learn best strategy (offline) to perform tailored control of the HVAC unit for any building types
- Eliminates the need to find and set distinct HVAC control strategies for each building

Use case 2: Sensitive Medical Data



Providing sensitive medical data for research

- Data required for most researches are sensitive in nature as it revolves around private individuals
- Resource-constrained medical devices like insulin-delivery devices, electric steam sterilizers, BP apparatus, etc., can use proposed algorithms to train offline using the sensitive data
- Transmit models instead of the actual medical data

Use case 3: Learning any Patterns



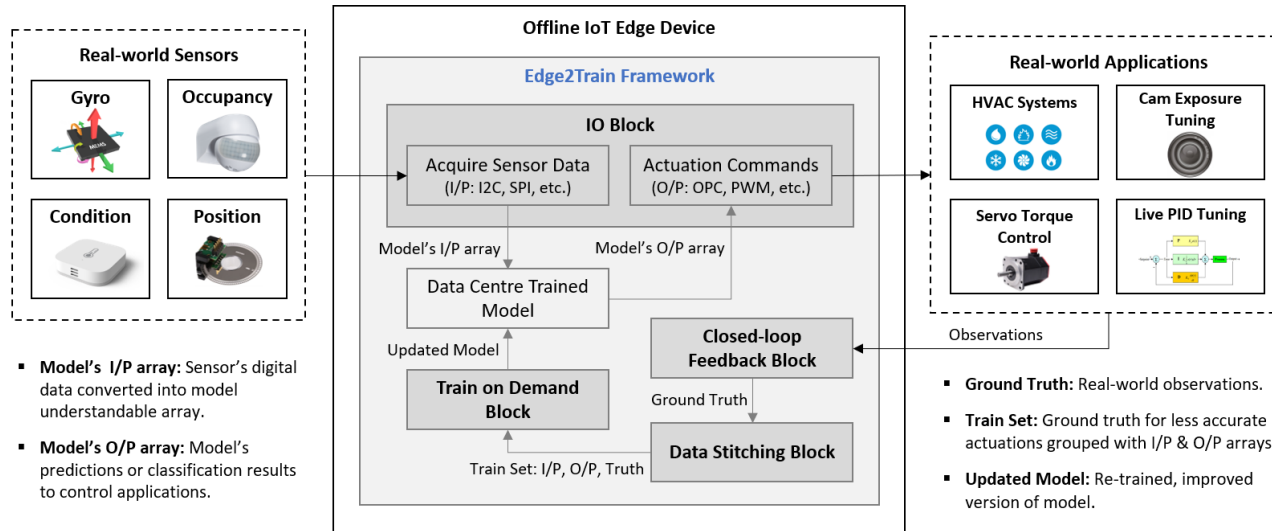
Use case: Learn to monitor vibrations on pump's crosshead, cylinder and frame. Generate alerts if anomaly patterns are detected

Use case: Learn to detect Leaks, monitor efficiency & consumptions



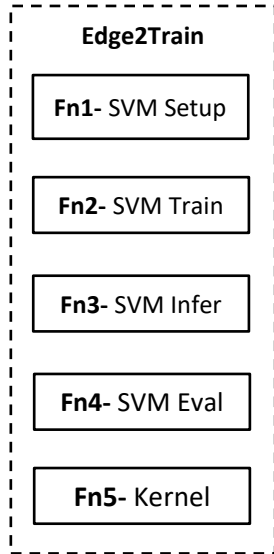
Self-learning the data patterns

- Proposed algorithms can train models on MCUs. Thus, even the tiny IoT devices can self-learn data patterns
 - ✓ Learn usual residential electricity consumption patterns and raise alerts in the event of unusual usage or overconsumption. Thus can reduce bills, detect leaks, etc.
 - ✓ Learn by monitoring the contextual sensor data corresponding to regular vibration patterns from the pump's crosshead, cylinder and frame
 - ✓ Generate alerts using the learned knowledge if anomaly patterns are predicted or detected



Architecture and components of Edge2Train: Training SVMs on IoT edge MCUs

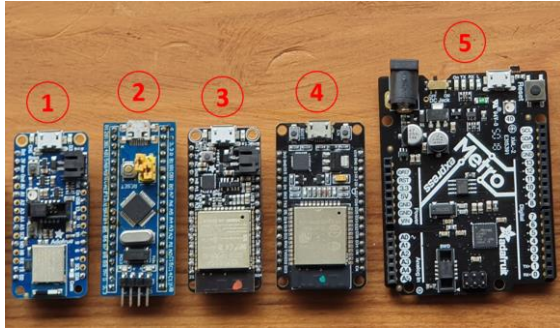
- Edge2Train functions **enables training SVMs on MCUs** using live data from their IoT use cases. Functions also enable on-board inference and model evaluations
- The Edge2Train blocks fuse with the device's IoT application to continuously **improve edge analytic results** by training using the evolving real-world data



Edge2Train
functions

- The five functions that constitute C++ Edge2Train library are
 - ✓ **Fn1 - SVM Setup:** Sets up parameters that tune the SVM
 - ✓ **Fn2 - SVM Train:** This function uses the local train sets to train SVMs on MCUs
 - ✓ **Fn3 - SVM Infer:** When sensor data is fed to this function, it calls the newly trained SVM, performs classifications, and sends results to the IoT application
 - ✓ **Fn4 - SVM Eval:** Evaluate the classification accuracy of the MCU-trained SVMs
 - ✓ **Fn5 - Kernel:** Produce implicit dot products of two values. On MCUs, it is used during training SVMs and while inferring using trained SVMs

Edge2Train Evaluation Setup

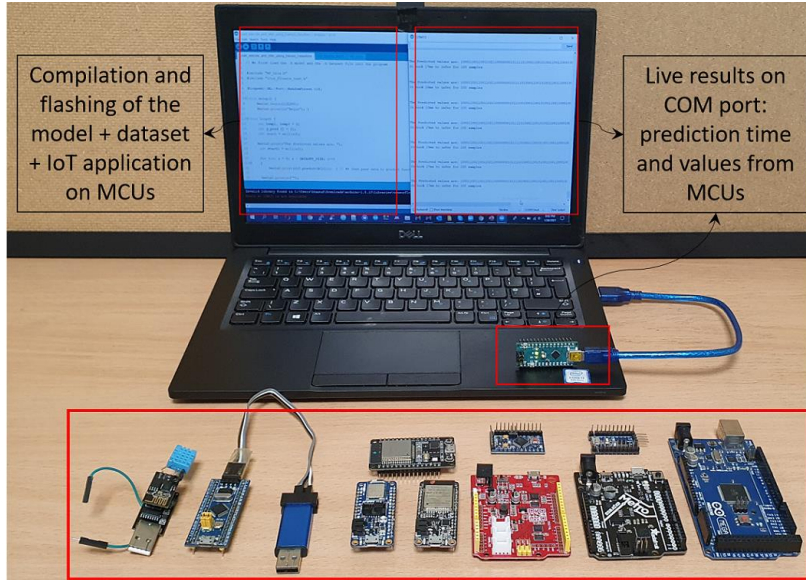


Board	MCU & Board Name	Specification				
		Bits	SRAM	Flash	Clock (MHz)	FP
#1	nRF52840 Adafruit Feather	32	256kB	1MB	64	✓
#2	STM32f103c8 Blue Pill	32	20kB	128kB	72	✗
#3 #4	Adafruit HUZZAH32, Generic ESP32	32	520kB	4MB	240	✓
#5	ATSAMD21G18 Adafruit METRO	32	32kB	256kB	48	✗

Popular open source MCU boards
chosen to evaluate Edge2Train

- **Hardware:** We evaluate Edge2Train by training SVMs on the shown MCU boards. Selected boards are
 - ✓ Popular, open source, globally available to purchase, low cost of 3 - 20 \$, widely used by DIY communities, such MCUs are used to design IoT devices
- **Datasets:** We used Iris flower (feature dimension of 4) and MNIST handwritten digits (feature dimension of 64)
 - ✓ Standard toy datasets widely used in ML community for initial algorithm evaluations
- **Comparison:** After training we compare the accuracy, power consumed and inference time of our Edge2Trained models with Python Scikit learn models on high-resource CPUs

Edge2Train Evaluation Setup

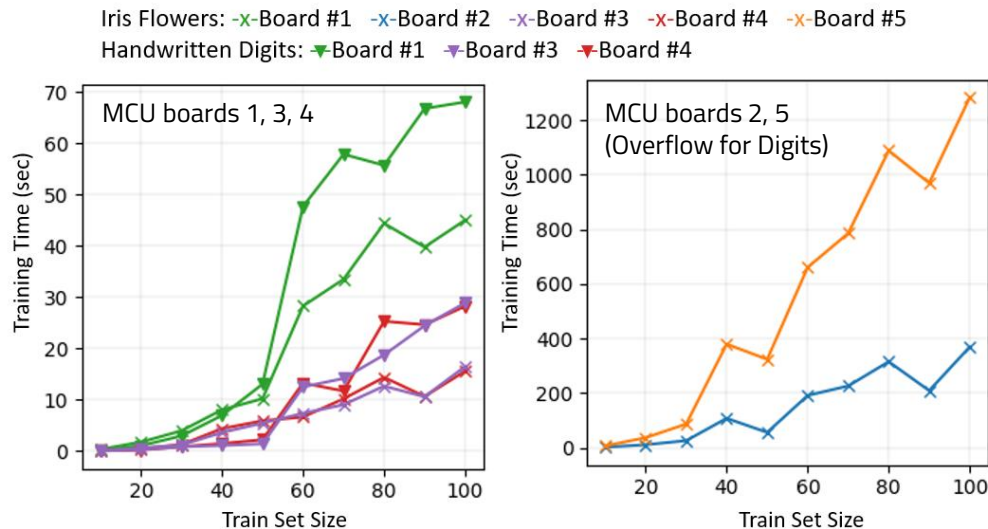


Experiment setup: Onboard SVM training and inference using Edge2Train

- We received the training and inference result from MCUs via Serial port. We report the following results, using which we perform analysis in the upcoming slides
 - ✓ Time and energy consumed to; Train SVMs on MCUs; Infer using thus trained SVMs on MCUs
 - ✓ Flash and SRAM consumed by Edge2Train on various MCU boards
 - ✓ Compare MCUs results (uses Edge2Train) with CPUs (uses Python scikit-learn) for the same datasets and tasks

Results: SVMs Train Time on MCUs

- The relationship between training time, training set size, and feature dimension

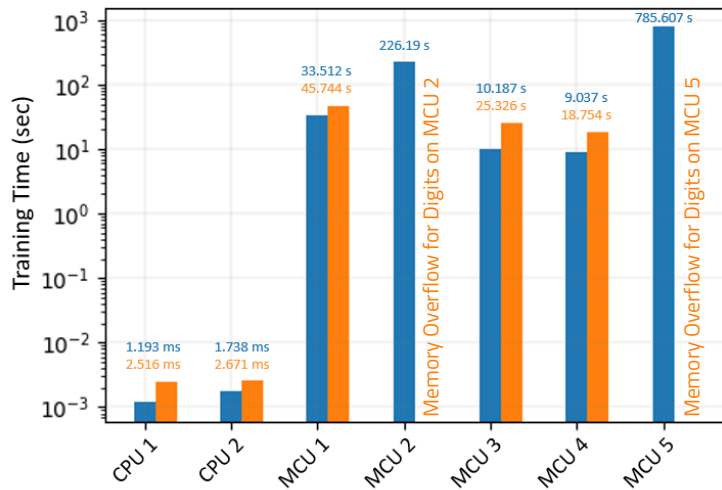


Training SVMs on MCUs using Edge2Train: Comparing training set size (number of data rows) vs training time (sec)

- ✓ ESP32 took 15.58 secs to train for Iris. 28.24 secs for 64 features MNIST digits
- ✓ Metro with 48 MHz clock and no FPS took 21.39 minutes for Iris (82x times slower than ESP32)
- ✓ Training time grows swiftly with the number of training samples
- ✓ Sudden peaks observed when Edge2Train consumes more time to find the optimal hyperplane
- ✓ Trough points are observed when the optimal hyperplane is found quickly

Results: Train Time Comparison

Figure 3.a. Time consumed by devices to train an SVM using: ■ Iris Flowers ■ Handwritten Digits



Comparing training time on MCUs (uses Edge2Train) vs CPUs (uses Python scikit-learn)

- The chosen CPUs have 1000x better specifications over MCUs
 - ✓ The train time for all 5 MCUs and 2 CPUs are shown
 - ✓ CPU1 and MCU4 is fastest in their respective classes
 - ✓ **CPU1 vs MCU4:** CPU1 is only 7.57 sec faster than MCU4 for Iris and 7.45 sec for MNIST digits datasets
- Although CPUs are faster, they cannot be used as edge devices due to their cost (CPU1 is 200x more costly than MCU4), form factor (5x more area), and energy consumption (7x times)
- Since billions of edge devices are MCU-based, it is feasible to train even at lesser speeds. Models can rather be trained on the edge using our framework

- MCU-based edge devices can use Edge2Train to continuously improve themselves for better edge analytics results by managing to understand continuously evolving real-world data
- Edge2Train Limitations
 - ✓ Memory overflow issues on MCUs when input dataset is larger than flash memory. This is due to the non incremental loading characteristic of the algorithm
 - ✓ Model train time is very large on 8 and 16 bit MCUs

- SVMs @ Edge2Train: <https://github.com/bharathsudharsan/Edge2Train>

main
1 branch
0 tags
Go to file
Code

bharath sudharsan Update README.md		sffcc58 on Aug 6	38 commits
E2T	Added code	6 months ago	
README.md	Update README.md	last month	
Setup.png	adding results	3 months ago	
Train_and_infer_energy_on_mcus_and...	fil	2 months ago	
Train_and_infer_time_on_mcus_and_c...	fil	2 months ago	
samples_vs_time.png	Update samples_vs_time.png	2 months ago	

☰ README.md

Support-Vector Machines (SVMs) Training and Inference on Microcontrollers (MCUs)

About

Code for paper: 'Edge2Train: a framework to train machine learning models (SVMs) on resource-constrained IoT edge devices'

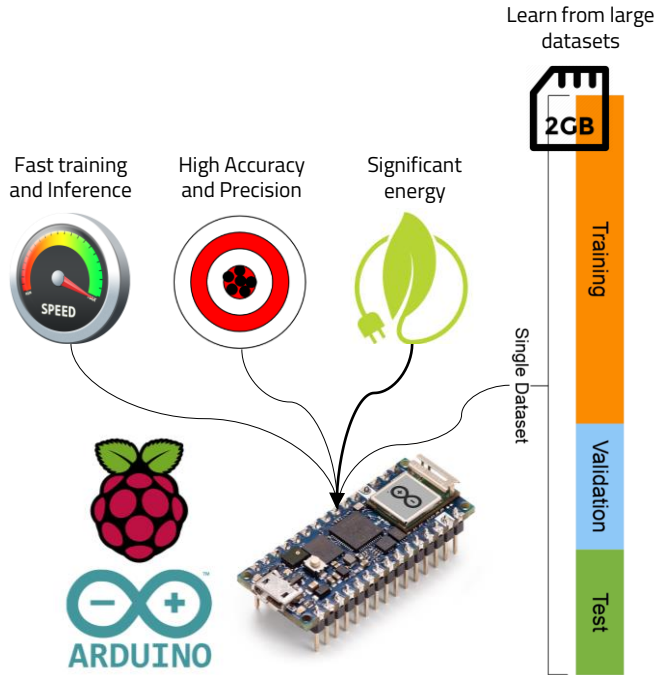
microcontroller
optimization
svm-training
online-learning
arm-cortex-m0
edge-computing
efficient-inference
arm-cortex-m4
iot-devices
tinyml

☐ README

Languages

C 90.9%

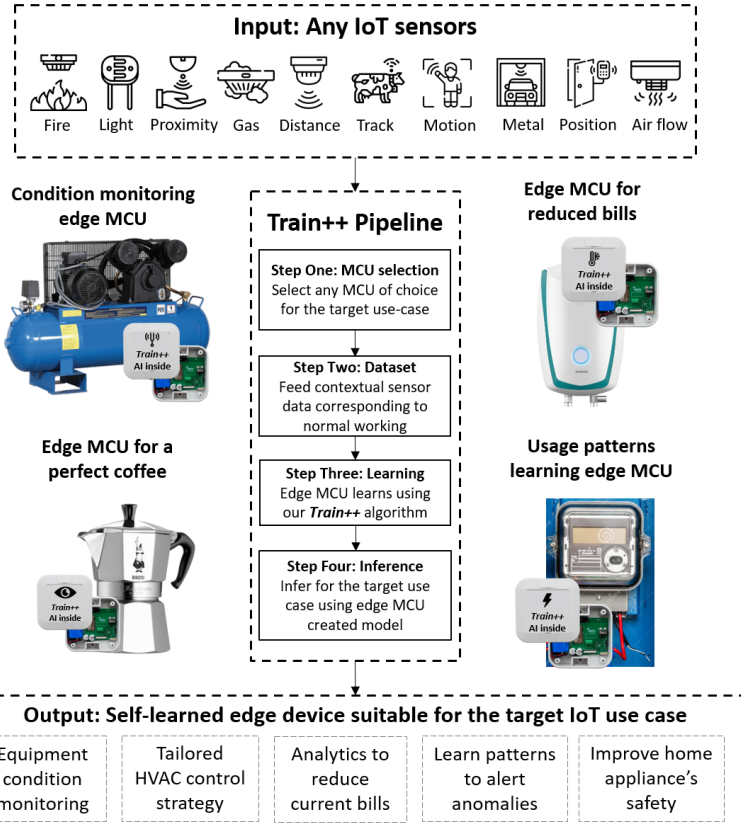
C++ 9.1%



Benefits of Training and Inference
on MCUs using Train++

- Train++ is designed for MCUs to perform on-board binary classifier model training, inference. Train++ algorithm main highlights/benefits
 - ✓ **Dataset size:** Incremental training characteristic enables training on limited Flash and SRAM footprints while also allowing to use full n -samples of the *high-dimensional* datasets
 - ✓ **Training speed and energy:** Achieves significant energy conservations due to its high-performance characteristics i.e., it trained and inferred at much higher speeds than other methods
 - ✓ **Performance:** Train++ trained classifiers show classification accuracy close to those of Python scikit-learn trained classifiers on high-resource CPUs

Train++: Pipeline



- Four-steps to use our Train++ algorithm to enable the edge MCUs to self-train offline for any IoT use cases
 - ✓ Select MCU of choice depending on the target use case
 - ✓ Contextual sensor data corresponding to normal working will be collected as the local dataset and used for training. The labels/ground truth for the collected training data rows shall be computed using our method from Edge2Train
 - ✓ Edge MCUs learn/train using our core Train++ algorithm
 - ✓ The resultant MCU-trained models can start inference over previously unseen data

Algorithm 1 Train++: A high-performance binary classifier training algorithm for MCU-based IoT Devices.

Inputs:

C : Positive parameter to control the influence of ξ .
 t : The dimension of time for real-time data inputs.
 x_t : Real-time sensor data inputs. Where $x_t \in \mathbb{R}^n$.
 y_t : Correct labels. Where $y_t \in \{-1, +1\}$.

Output:

w_t : Incrementally learned weights.

Receive live data stream, represented as in Eqn (1).

function MCUTrain (x_t, y_t, t, C)

for $t = 0$ to setsize **do**

Predict \hat{y}_t for every x_t . Use $\text{sign}(x_t \cdot w_t)$.

Compute confidence score of prediction. Use $|x_t \cdot w_t|$.

Compute margin at t . Use $y_t(x_t \cdot w_t)$.

Show original label y_t to this algorithm.

if (\hat{y}_t equals to y_t) **then**

Prediction was correct. $y_t = \text{sign}(x_t \cdot y_t)$.

Predict again with a higher confidence score.

if ($\text{sign}(x_t \cdot y_t)$ lesser than 1) **then**

Suffer an instantaneous loss. Use Eqn (2).

if ($\text{sign}(x_t \cdot y_t)$ exceeds 1) **then**

Loss becomes 0.

else

Wrong prediction. Loss becomes $1 - y_t(x_t \cdot w_t)$.

end if

Incrementally learn w_t . Use Eqn (4).

Store the learned weights w_t .

end for

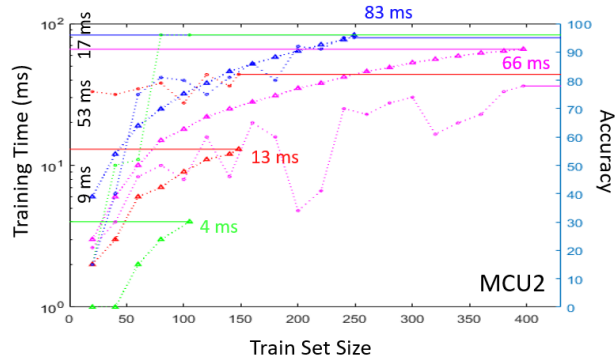
- The full Train++ algorithm is shown. Characteristics
 - ✓ Highly resource-friendly and low complexity design
 - ✓ Implementation less than 100 lines in C++
 - ✓ Does not depend on external or third-party libraries
 - ✓ Executable even by 8-bit MCUs without FPU, KPU, APU support
 - ✓ Incremental data loading and training characteristics
 - ✓ Compatible with datasets that have high feature dimensions
 - ✓ Keeps reading live data -> incrementally learns from it -> discards data (no storing required)

Dataset#: Name - feature dimension			
Datasets	D1: Iris Flowers [34] - 4	D5: Banknote [35] - 5	
	D2: Heart Disease [36] - 13	D6: Survival [37] - 3	
	D3: Breast Cancer [38] - 30	D7: Titanic [39] - 11	
	D4: MNIST Digits [40] - 64		
MCU#		Name	Specs: processor flash, SRAM (kB), clock (MHz)
MCU boards	1	nRF52840 Feather	Cortex-M4, 1MB, 256, 64
	2	STM32f10 Blue Pill	Cortex-M0 128kB, 20, 72
	3	Adafruit HUZZAH32	Xtensa LX6,
	4	Generic ESP32	4MB, 520, 240
	5	ATSAMD21 Metro	Cortex-M0+, 256kB, 32, 48

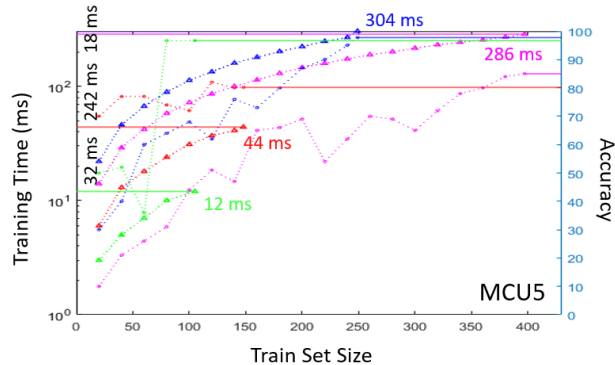
Datasets and MCUs used for Train++ evaluation

- Using Train++, for datasets D1-D7, we train a binary classifier on MCUs 1-5. This multiple datasets and MCUs based extensive experimental evaluation aims to answer
 - ✓ Is Train++ compatible with different MCU boards, and can it train ML models on MCUs using various datasets with various feature dimensions and sizes?
 - ✓ Can Train++ load, train, and infer using high features and size datasets on limited memory MCU boards that have low hardware specification and no FPU, APU, KPU support?

Results: Train Time



Iris Flowers: ▲ Train Time ● Accuracy
Heart Disease: ▲ Train Time ● Accuracy



Breast Cancer: ▲ Train Time ● Accuracy
Handwritten Digits: ▲ Train Time ● Accuracy

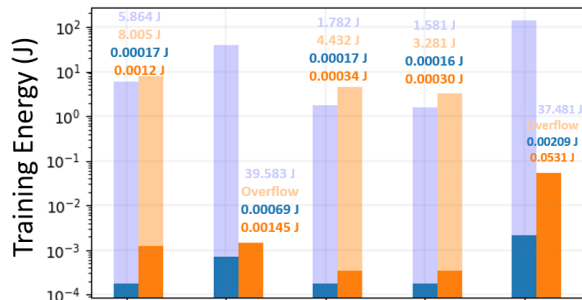
- Training models on MCUs using Train++: Comparing training set size vs training time and accuracy for selected datasets
 - ✓ We show results only for MCUs 2, 5 as other MCUs trained in 2 ms
 - ✓ The gap in y-axis is difference in train time between datasets
 - ✓ Train time grows almost linearly with samples for all datasets
 - ✓ For the Iris dataset MCU2 only took 4 ms to train on 105 samples
 - ✓ It took 83 ms to train on the Digits dataset
 - ✓ MCU5 is slowest since it only has a 48 MHz clock, does not have FPU support. Hence it took 12 ms to train on 105 samples of the Iris dataset (3x times slower than MCU2) and 304 ms to train on the Digits dataset (3.6x times slower than MCU2)

- **Memory:** Flash and SRAM consumption is calculated during compilation by the IDE
 - ✓ For MCU1, Iris dataset and Train++ in total used only 4.1 % of Flash and 3.4 % SRAM. For Digits, the same MCU1 requires 6.54 % and 29.93 %
 - ✓ When using Edge2Train, for MCUs 2, 5, we cannot train using the Digits dataset because SRAM overflowed by +52.0 kB and +63.62 kB. Similarly for Heart Disease both the Flash and SRAM requirements exceed the MCU's capacity
 - ✓ Even after overflow on MCU2, Train++ incremental training characteristics could load the dataset incrementally and complete training in 83 ms
 - ✓ Similarly, even after overflow on MCU5, Train++ was able to load the entire dataset incrementally and train in 304 ms
- **Inference time:** For the selected datasets, on MCUs 1-5, Train++ method infer for the entire test set in lesser time than the Edge2Train model's unit inference time

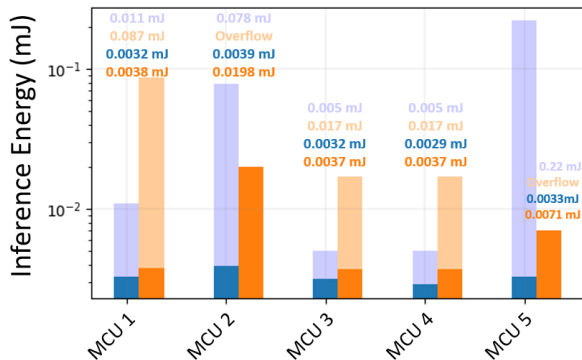
- **Accuracy:** The highest onboard classification accuracy is 97.33 % for the Iris (D1), 82.08 % for Heart Disease (D2), 85.0 % for Breast Cancer (D3), 98.0 % for Digits dataset (D4)
- **Accuracy comparison:** When comparing the accuracy of Train++ trained models with Edge2Train trained models
 - ✓ For the same Iris dataset, the accuracy improved by 7.3% and by 5.15% for the Digits dataset. This improvement is because our training algorithm enabled incremental loading of the full dataset
 - ✓ Other algorithms like SVMs work in batch mode, requiring full training data to be available in the limited MCU memory, thus sets an upper bound on the train set size
 - ✓ Train++ trained models achieved overall improved accuracy compared to the Edge2Train models, which were trained with limited data (unable to load full dataset due to memory constraints)

Results: Edge2Train vs Train++

a. Training energy consumed by MCUs.



b. Unit inference energy consumed by MCUs.



Edge2Train: Iris Flowers ■ Handwritten Digits ■

Train++: Iris Flowers ■ Handwritten Digits ■

- From the shown Figure (y-axis in base-10 log scale) it is apparent that Train++ consumed
 - ✓ 34000 - 65000x times less energy to train
 - ✓ 34 - 66x times less energy for unit inference
- For a given task that needs to be completed by using the same datasets on the same MCUs, Train++ achieved such significant energy conservations due to its high-performance characteristics (i.e., it trained and inferred at much higher speeds)
- When Train++ is used, MCUs can perform onboard model training and inference at the lowest power costs, thus enabling offline learning and model inference without affecting the IoT edge application routine and operating time of battery-powered devices

- Summary of Train++ benefits
 - ✓ The proposed method reduces the onboard binary classifier training time by $\approx 10 - 226$ sec across various commodity MCUs
 - ✓ Train++ can infer on MCUs for the entire test set in real-time of 1 ms
 - ✓ The accuracy improved by 5.15 - 7.3 % since the incremental characteristic of Train++ enabled loading of full n-samples of high-dimensional datasets even on MCUs with only a few hundred kBs of memory
 - ✓ Train++ is compatible with various MCU boards and multiple datasets
 - ✓ Across various MCUs, Train++ consumed $\approx 34000 - 65000$ x times less energy to train and consumed $\approx 34 - 66$ x times less energy for unit inference

- Incremental ultra-fast @ Train++: https://github.com/bharathsudharsan/Train_plus_plus


master ▾

1 branch

0 tags

Go to file

Code ▾

 bharathsudharsan Update README.md 88b54dd on Aug 2 71 commits

Train_plus_plus	readme with results	4 months ago
LICENSE	Initial commit	12 months ago
README.md	Update README.md	2 months ago
energy_comparison_for_train_and_inf...	image resize	4 months ago
setsize_vs_train_time_and_accuracy.png	image resize	4 months ago

☰ README.md

Resource-friendly, High-performance ML Classifier Training and Inference on Arduino MCUs

About

Repo and code of the IEEE UIC paper: Train++: An Incremental ML Model Training Algorithm to Create Self-Learning IoT Devices

machine-learning

microcontroller

optimization

esp32

adafruit

arduino-ide

online-learning

stm32f103

incremental-learning

edge-computing

classifier-training

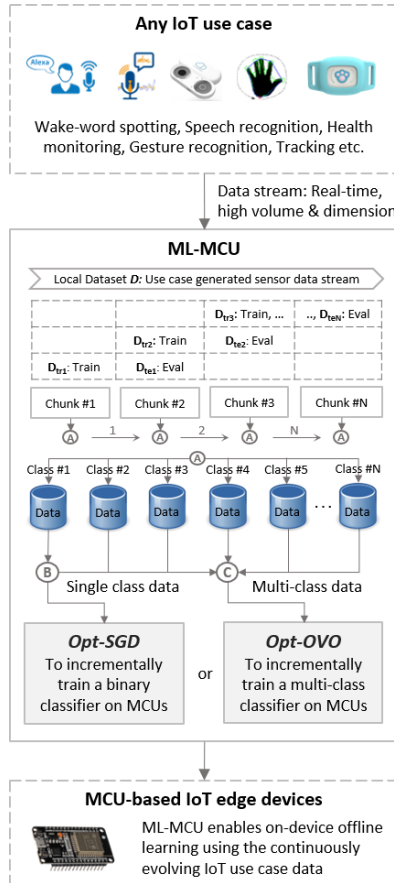
armcortexm4

armcortexm0

📖 Readme

📄 MIT License

Languages

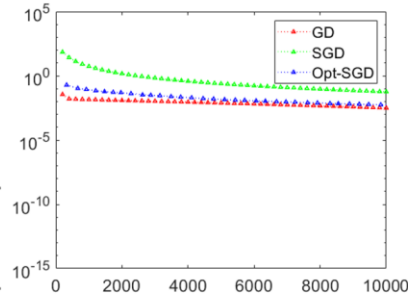


- Novelty: Opt-SGD and Opt-OVO algorithms
 - ✓ **Optimized-Stochastic Gradient Descent (Opt-SGD):** To enable high performance, resource-friendly binary classifiers training on small edge MCUs. This method combines benefits from both Gradient Descent (GD) and Stochastic Gradient Descent (SGD) thus, inheriting the stability of GD while retaining the work-efficiency of SGD
 - ✓ **Optimized One-Versus-One (Opt-OVO):** To the best of our knowledge, this is the first novel algorithm to enable multi-class classifiers training on MCUs. Opt-OVO archives reduced computation by identifying and removing base classifiers that lack significant contributions to the overall multi-class classification result

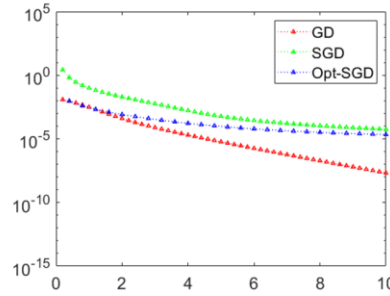
ML-MCU Results: Convergence

- Comparing convergence behavior of Opt-SGD with GD, SGD

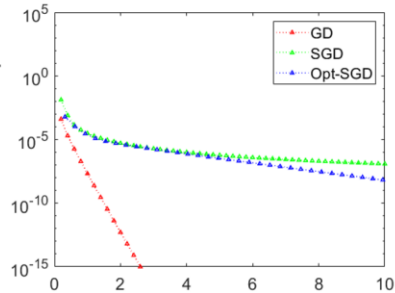
- ✓ Convergence of GD and Opt-SGD is linear. SGD shows sub-linear pattern
- ✓ As data increases, inefficiency of GD increases since to update the model weights for each iteration, the algorithm needs to pass through the entire dataset
- ✓ The convergence of SGD is very slow, whereas Opt-SGD was capable to reach high precision despite its stochastic characteristic
- ✓ For a very large train set, our method is less efficient than SGD. But practically we don't train using large data on MCUs. Anyways ML-MCU initially splits the data stream into small data chunks, for which our algorithm performs better than GD (higher precision) and faster than SGD



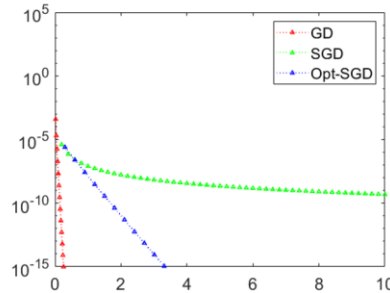
a. X-axis: Gradient execution count



b. X-axis: Gradient execution count $\times 10^4$



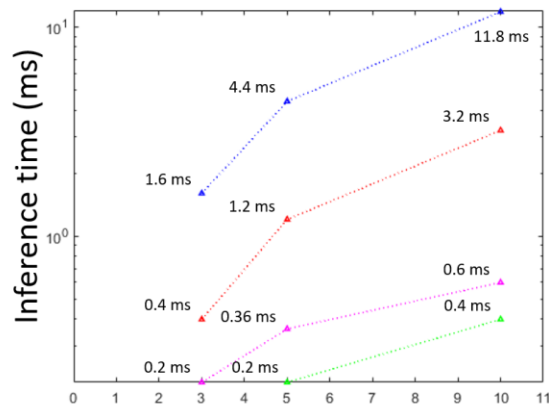
c. X-axis: Gradient execution count $\times 10^5$



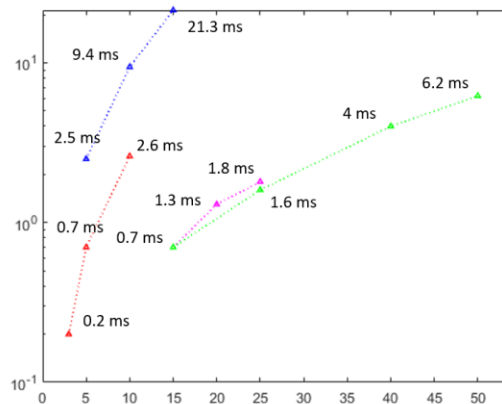
d. X-axis: Gradient execution count $\times 10^6$

ML-MCU Results: Infer Time

MCU #1: ▲ Unit Infer time. MCU #2: ▲ Unit Infer time.
MCU #3: ▲ Unit Infer time. MCU #4: ▲ Unit Infer time.



a. X- axis: Digits dataset:
Class count (0 - 10)



b. X-axis: Australian Sign Language
signs dataset: Class count (0 - 50)

- Real-time unit inference in 11.8 ms, even on the slowest MCU4
- The fastest MCU3 was able to infer for a 50 class input in 6.2 ms
- Overall, it is apparent that our Opt-OVO trained classifiers perform onboard unit inference for multi-class data in super real-time, within a second, across various MCUs

Inferring for multi-class data on MCUs: Class size (3 to 50) vs inference time of the onboard Opt-OVO trained classifiers

- Multi-class @ ML-MCU: <https://github.com/bharathsudharsan/ML-MCU>

master ▾

1 branch

0 tags

Go to file

Code ▾

bharath sudharsan Update README.md

ffce290 15 days ago 62 commits

Opt-OVO	code updates	12 months ago
Opt-SGD	code updates	12 months ago
LICENSE	Initial commit	12 months ago
README.md	Update README.md	15 days ago
multiclass_inference_results.png	results	2 months ago
multiclass_training_results.png	results	2 months ago
opt-sgd_results.PNG	results_add	2 months ago

README.md

Multi-class ML Classifier Training and Real-time Inference on Arduino MCUs

About

Code for IoT Journal paper title 'ML-MCU: A Framework to Train ML Classifiers on MCU-based IoT Edge Devices'

machine-learning

microcontroller

optimization

gradient-descent

online-learning

incremental-learning

edge-computing

classifier-training

sgd-optimizer

armcortexm4

armcortexm0

tinymml

Readme

MIT License

Languages

C 97.5%

C++ 2.5%

Presented algorithms that enable a range of MCUs that have limited Flash, SRAM, no FPU, FFT, KPU to re-train themselves locally and offline

- ✓ Edge2Train to enable onboard resource-friendly training of SVM models
- ✓ Train++ for ultra-fast incremental onboard classifier training and inference
- ✓ ML-MCU to train up to 50 class ML classifiers on a \$ 3 device ESP32 MCU board

Confirm

Smart Manufacturing

Confirm
Smart Manufacturing



Contact: Bharath Sudharsan
Email: bharath.sudharsan@insight-centre.org

www.confirm.ie


Science
Foundation
Ireland For what's next