tinyML® Summit

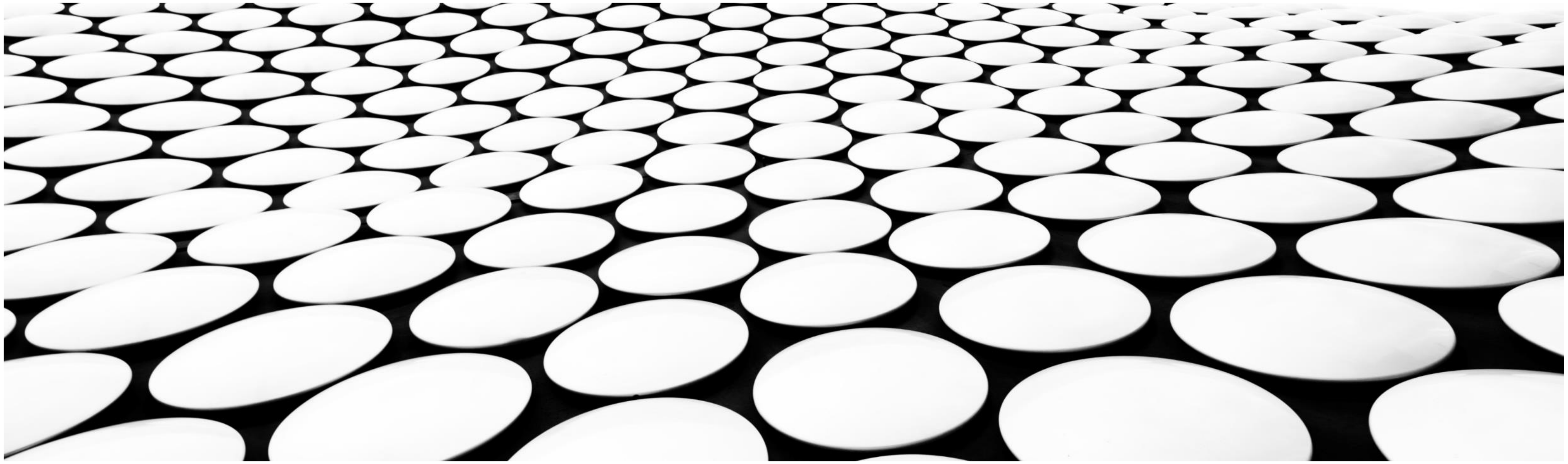
Miniature dreams can come true…

March 28-30, 2022 | San Francisco Bay Area

www.tinyML.org

AUTOMATED MACHINE LEARNING UNDER MODEL'S DEPLOYABILITY ON TINY DEVICES

ANTONIO CANDELIERI – UNIVERSITY OF MILANO-BICOCCA – DEPARTMENT OF ECONOMICS, MANAGEMENT AND STATISTICS



HYPERPARAMETER OPTIMIZATION (HPO)

It is the most well-known (and expensive) task!

- The ML algorithm is chosen a priori
- Let's suppose it has n hyperparameters $\gamma_1, \dots, \gamma_n$ with domains $\Gamma_1, \dots, \Gamma_n$
- The so-called "search space" $\Gamma \subseteq \Gamma_1 \times \cdots \times \Gamma_n$

- HPO is aimed a finding: $\boldsymbol{\gamma}^* \in \underset{\boldsymbol{\gamma} \in \Gamma}{\operatorname{argmin}} \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}\left(A_{\boldsymbol{\gamma}}, \mathcal{D}_{train}^{(i)}, \mathcal{D}_{valid}^{(i)}\right)$

with $\mathcal{L}\left(A_{\boldsymbol{\gamma}}, \mathcal{D}_{train}^{(i)}, \mathcal{D}_{valid}^{(i)}\right)$ a loss-function, averaged on $\boldsymbol{k}$ **fold-cross validation**

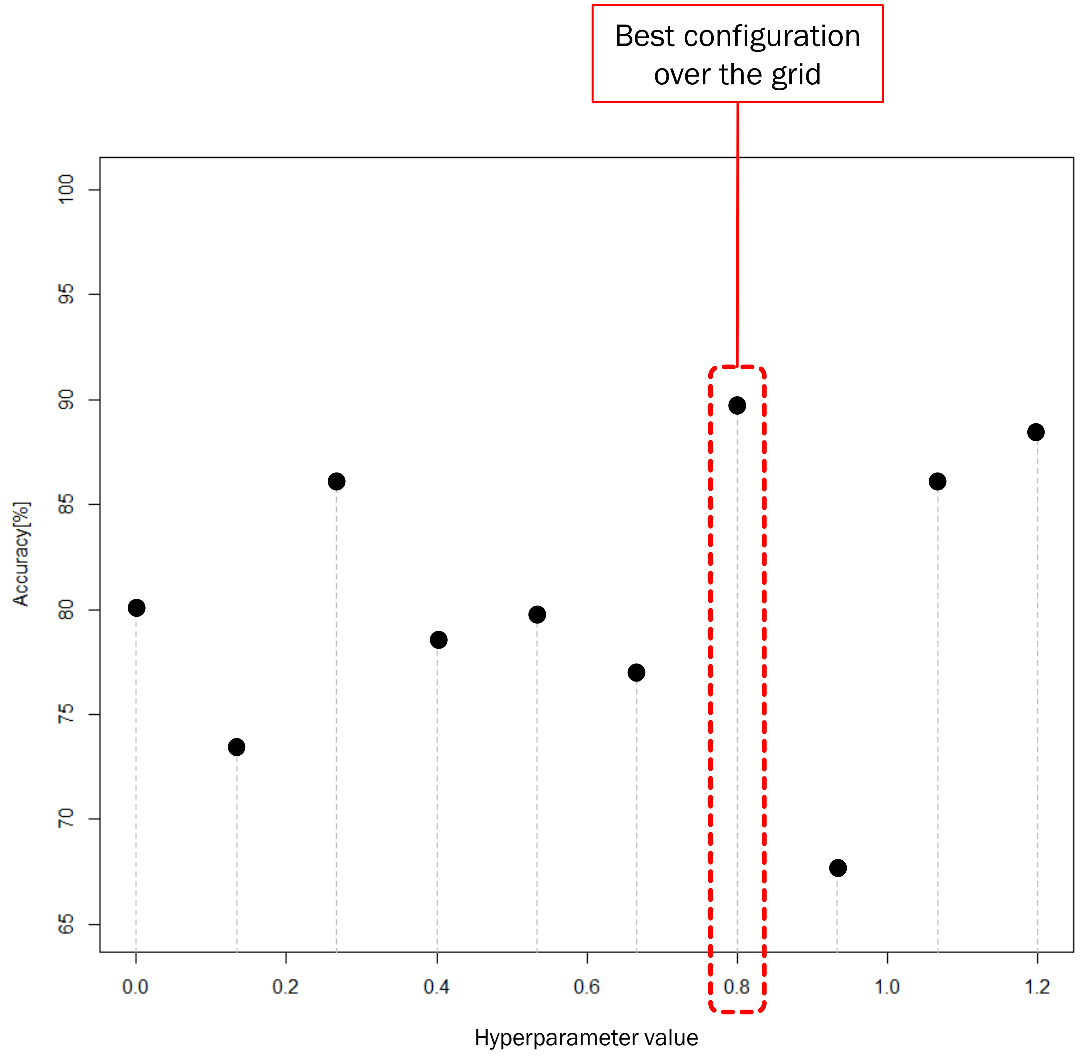
Thus, it is black-box and rexpensive, i.e., it equires to train and validate each $A_{\boldsymbol{\gamma}}$

GRID SEARCH

Grid Search is the simplest hyperparameter search method:

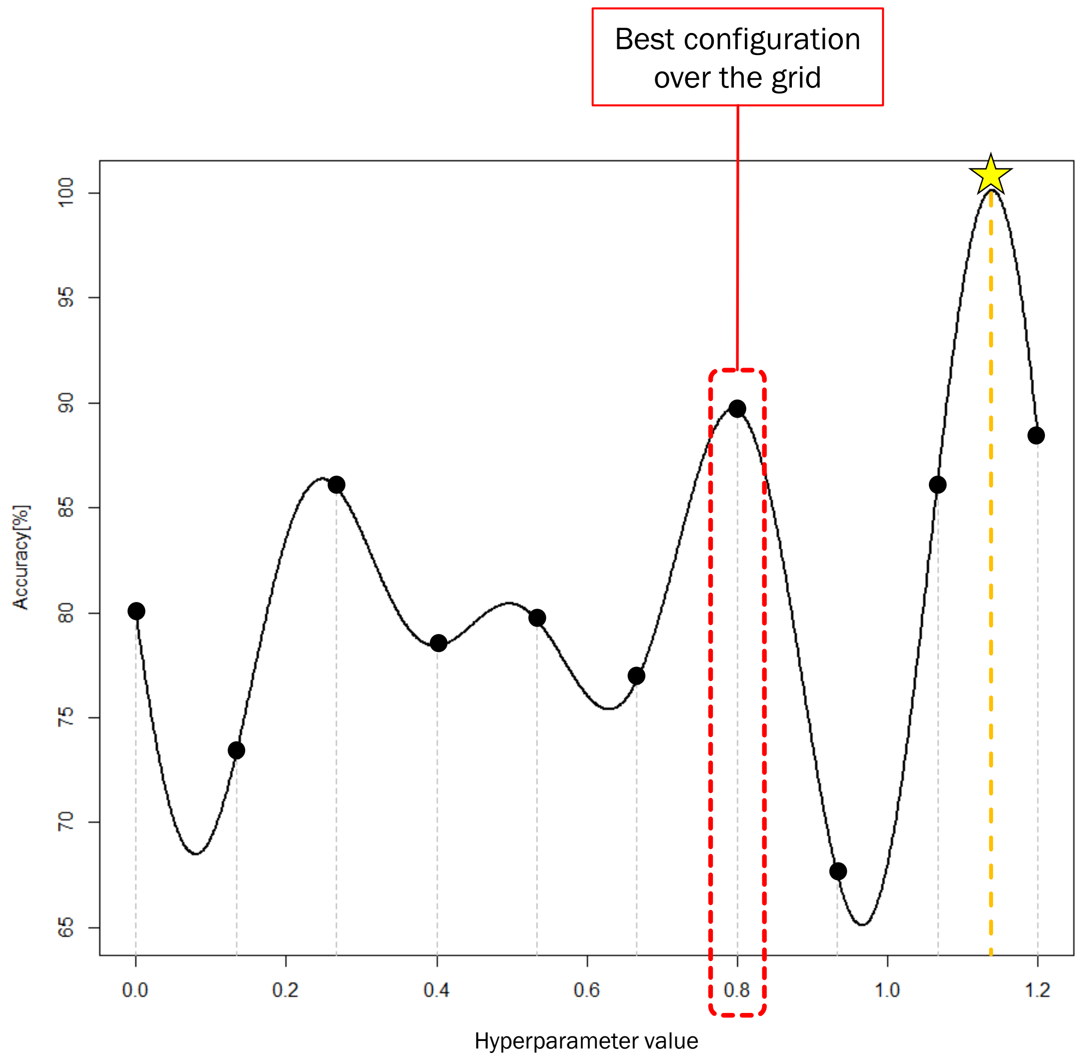
- easy to implement
- embarassingly parallel

On the right: we wanto to search for the hyperparameter value which maximizes the ***Accuracy [%] on k-fold cross validation***



DESPITE ITS SIMPLICITY, GRID SEARCH IS HIGHLY INEFFICENT

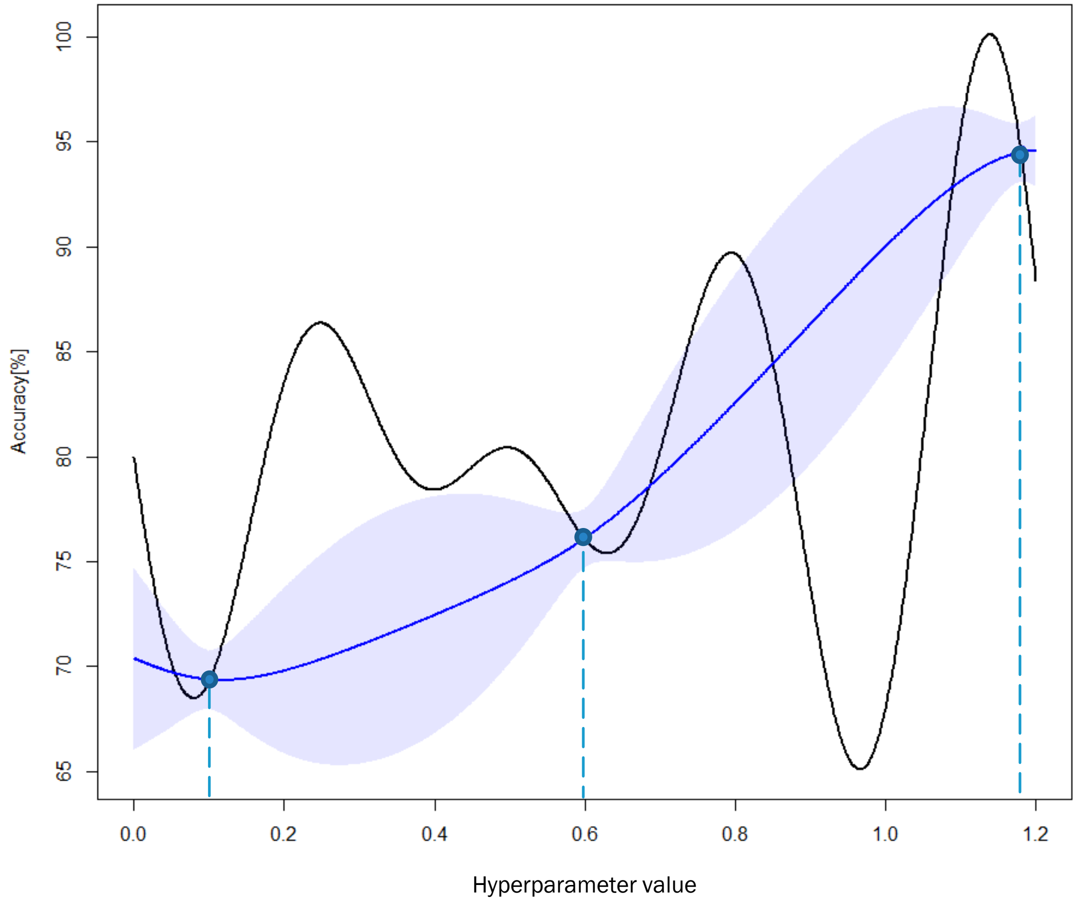
- Let's suppose now to know the value of the Accuracy over the entire search space…
- … it is clear that the configuration identified through Grid Search is far away from the omptimum!
- The question is: *"might we find a better solution by using the same number of trials?"*



BAYESIAN OPTIMIZATION IN A NUGGET! (A TOY EXAMPLE)

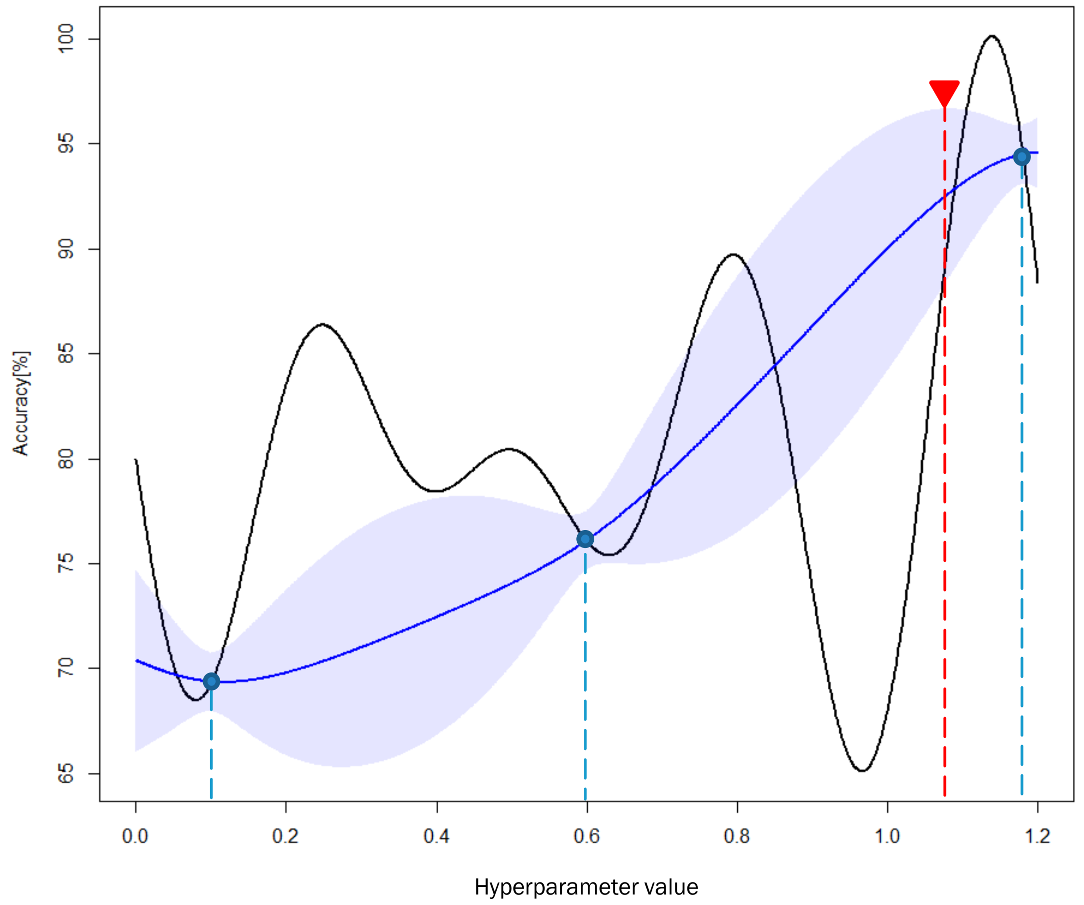
- Choose 3 random values for the hyperparameter and observe the associated Accuracies
- Fit a **probabilistic regression model** (e.g., a **Gaussian Process** – GP) to approximate the Accuracy over unseen values of the hyperparameter
- A probabilistic regression model provides both a **prediction** (solid blue line) and the associated **uncertainty** (blue shaded area)

- <u>Uncertainty is the key</u>!
- If we consider the prediction only, the best expected value for the validation loss would be already achieved…



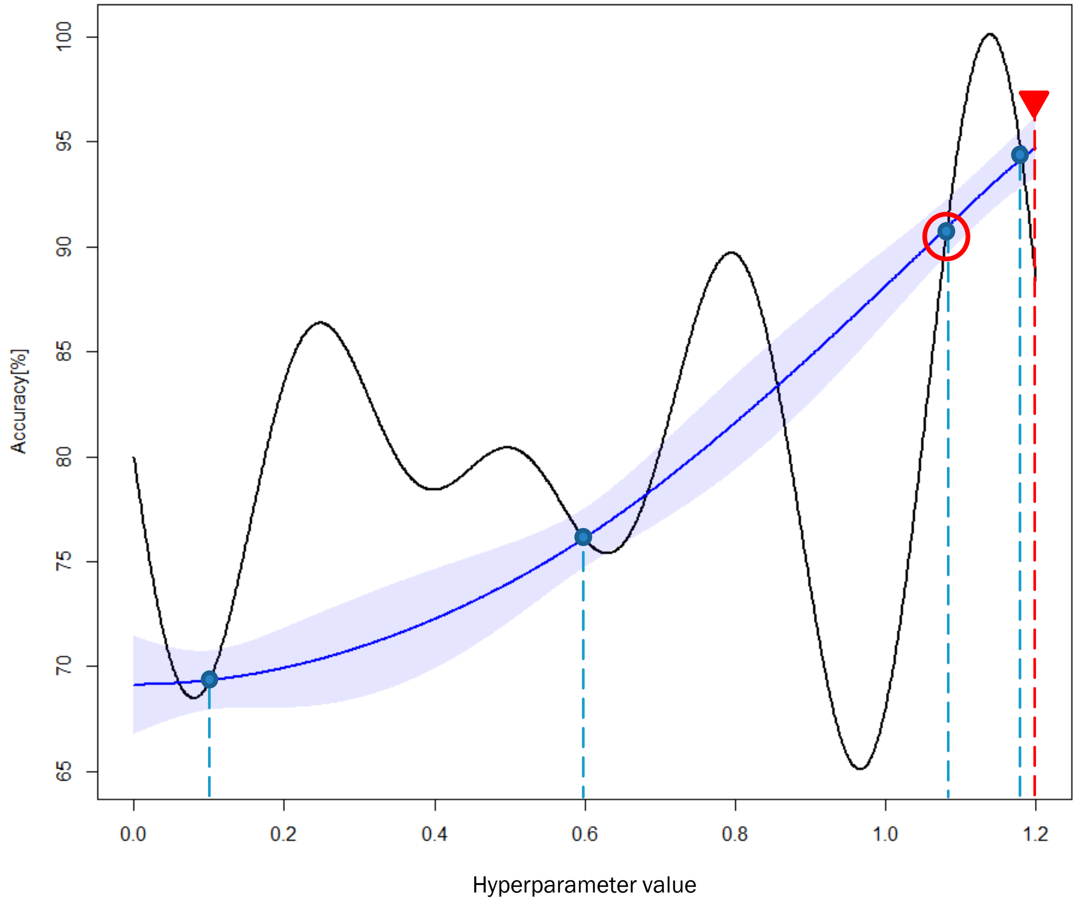
BAYESIAN OPTIMIZATION IN A NUGGET! (A TOY EXAMPLE)

- Instead, let's be optimistic in front of the uncertainty!
- The upper bound of the shaded area represents the most optimistic estimation for Accuracy with respect to the hyperparameter's value
- This selection mechanism is known as *Upper Confidence Bound*



BAYESIAN OPTIMIZATION IN A NUGGET! (A TOY EXAMPLE)

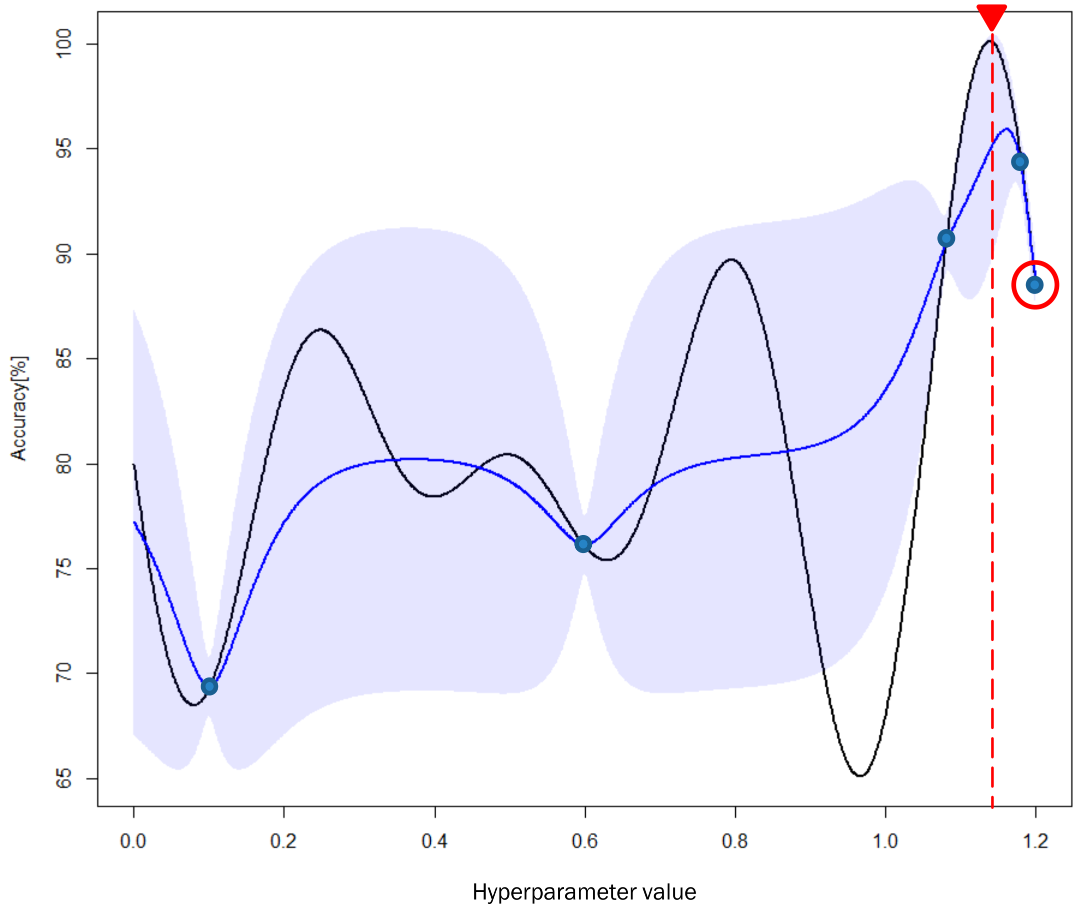
- Observe the Accuracy for the new hyperparameter value and update the probabilistic regression model
- In this case the approximation is not changed so much, but uncertainty is reduced
- We use again Upper Confidence Bound to select the next promising configuration



BAYESIAN OPTIMIZATION IN A NUGGET! (A TOY EXAMPLE)

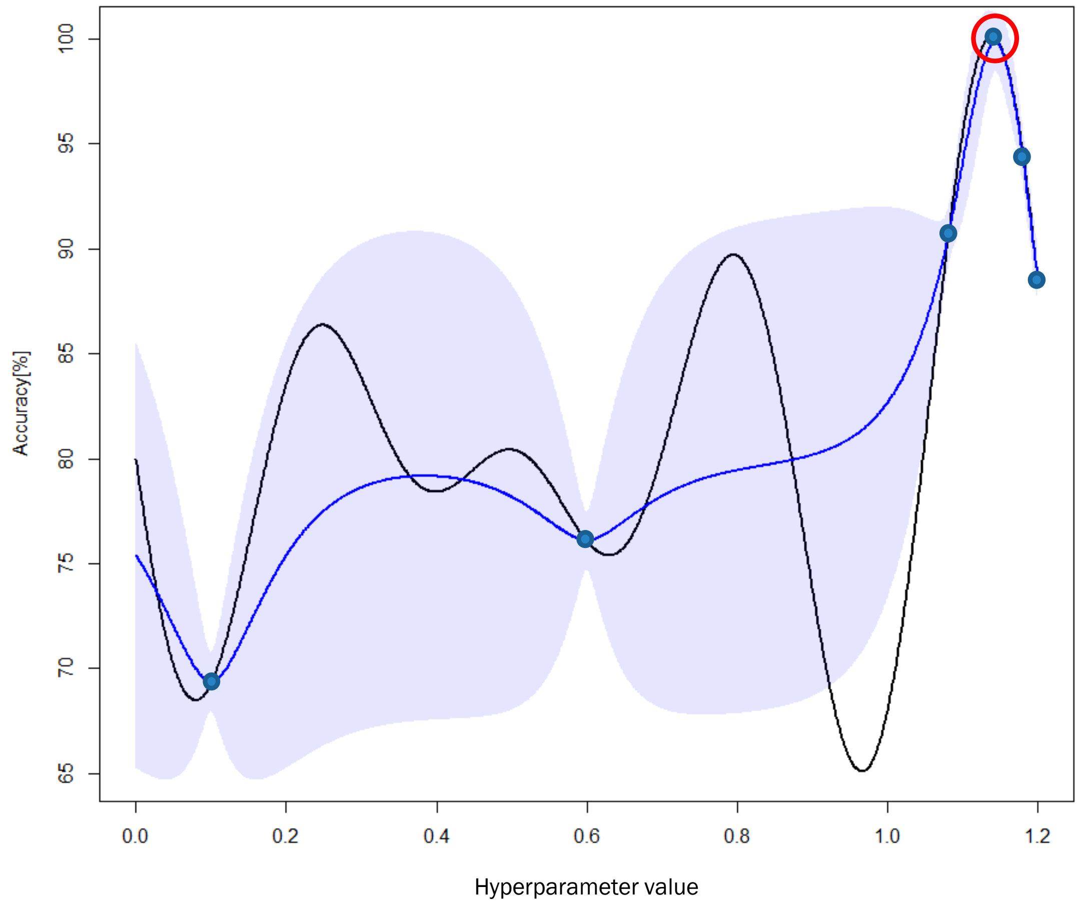
- In this case, the observed Accuracy leads to a significant change in both the prediction and the uncertainty of the probabilistic regression model

- Again, we can use Upper Confidence Bound to make our next choice…



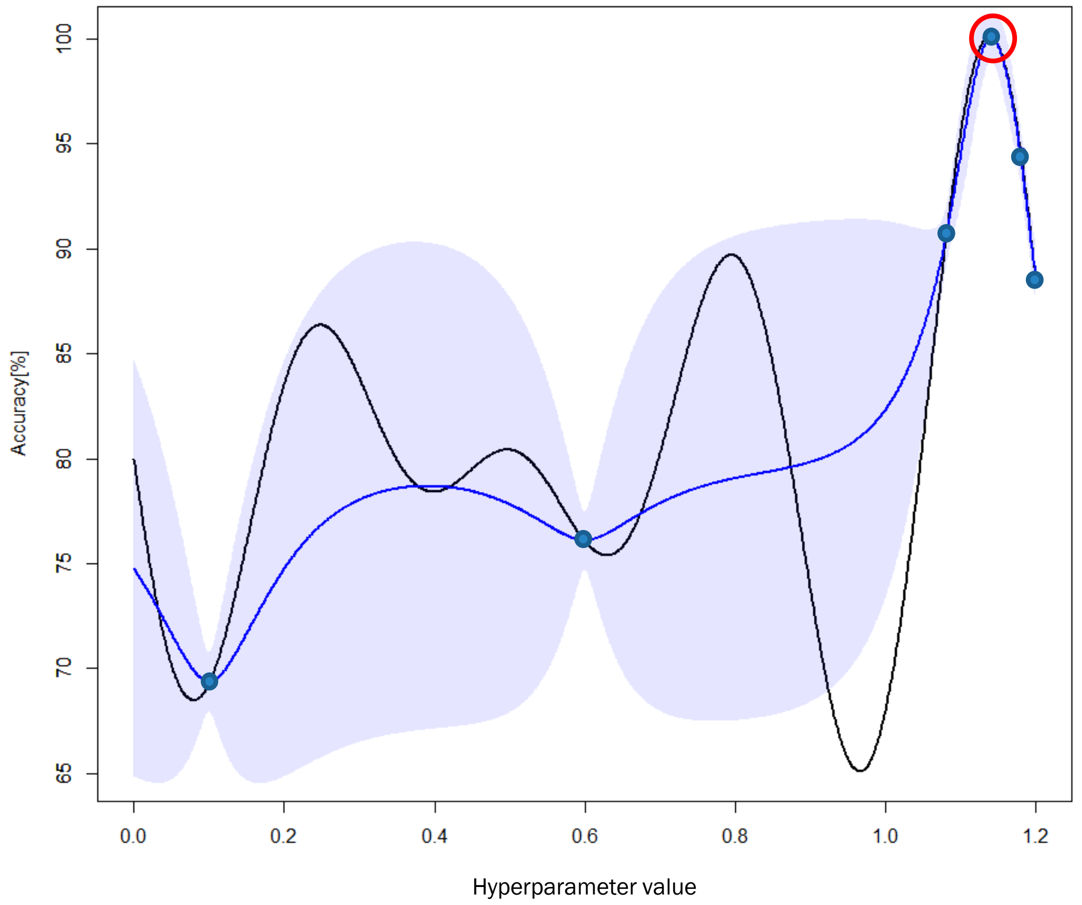
BAYESIAN OPTIMIZATION IN A NUGGET! (A TOY EXAMPLE)

- … and after 6 trials we are really close to the actual optimum!
- If we have other trials we can still iterate…



BAYESIAN OPTIMIZATION IN A NUGGET! (A TOY EXAMPLE)

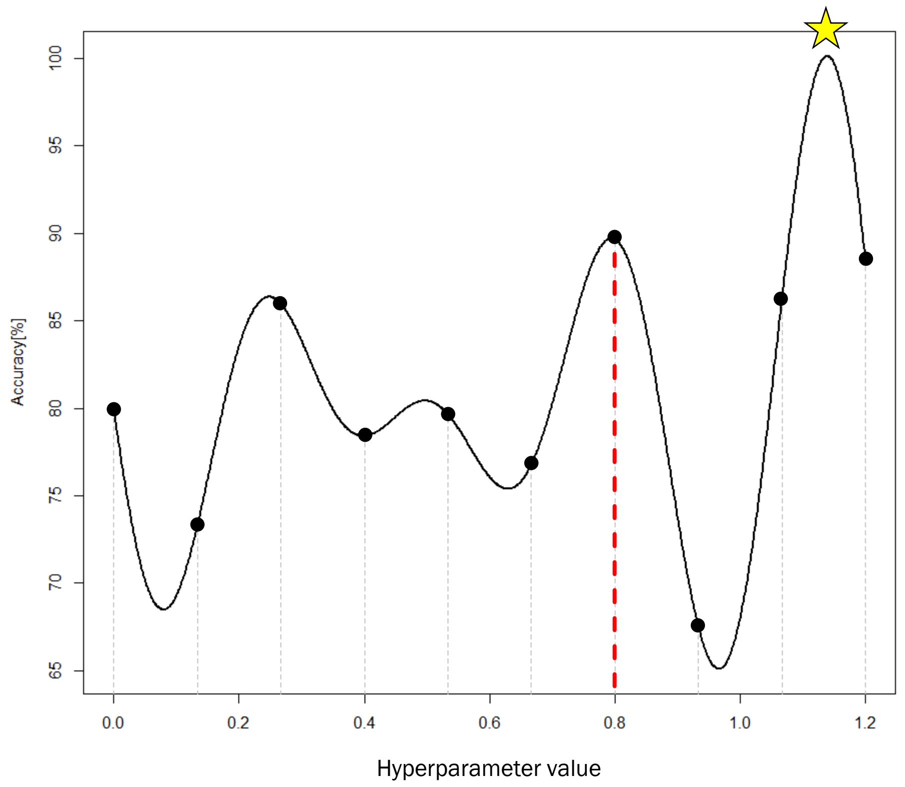
- Getting closer…
- …closer…
- …closer…
- … and closer!

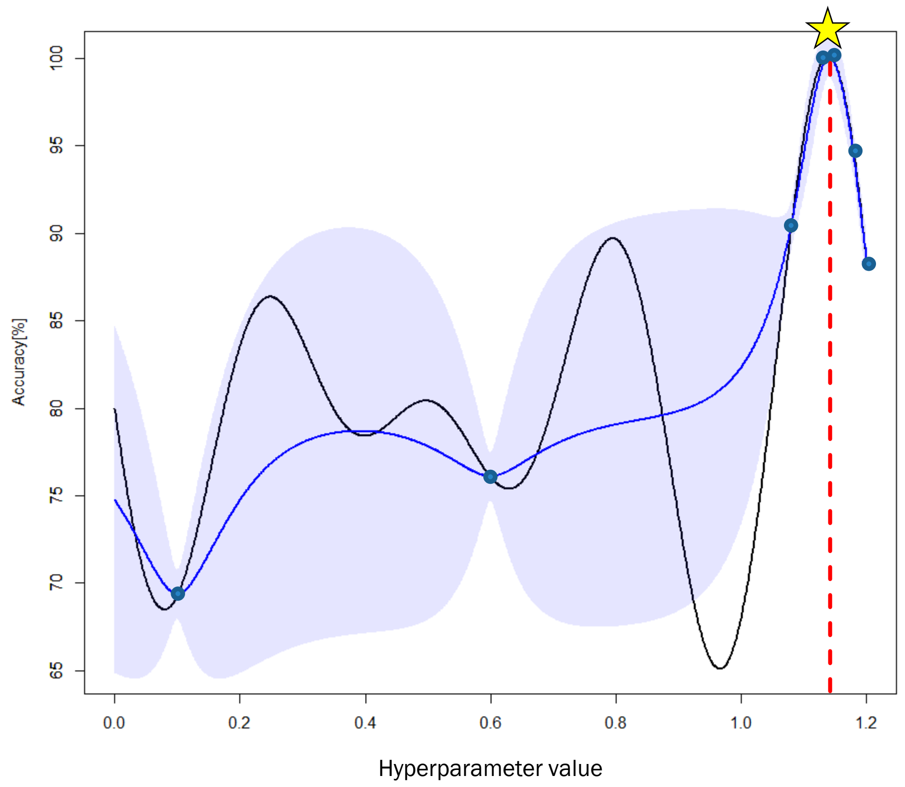


LET'S COMPARE GRID SEARCH WITH BAYESIAN OPTIMIZATION

The advantages offered by BO are definitely clear!

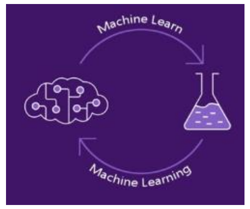
For this reason, BO is the standard method for Automated Machine Learning (AutoML)



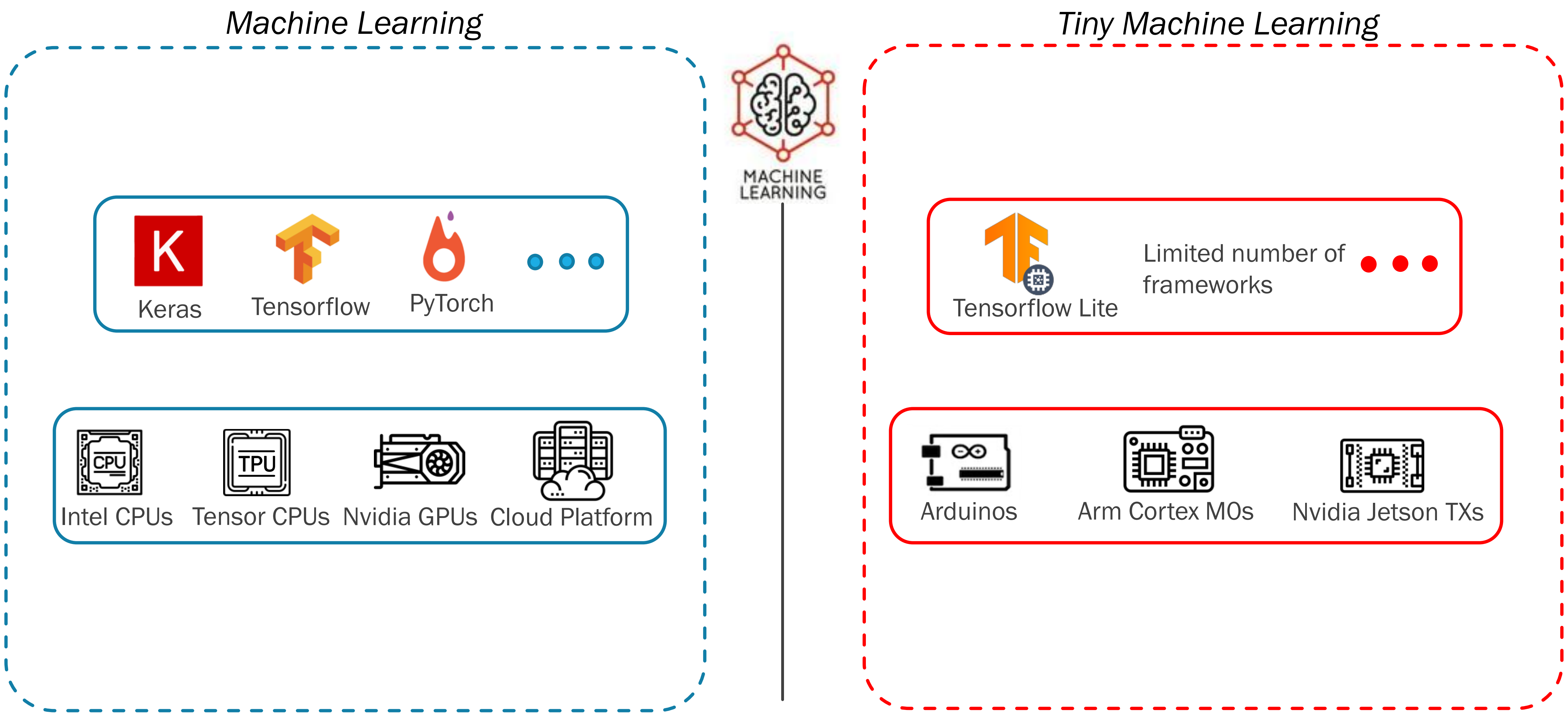


AUTOML AND NAS SOLUTIONS

- Google AutoML table (beta)
- Amazon Sage Maker
- Microsoft Research AutoML (it is a team)



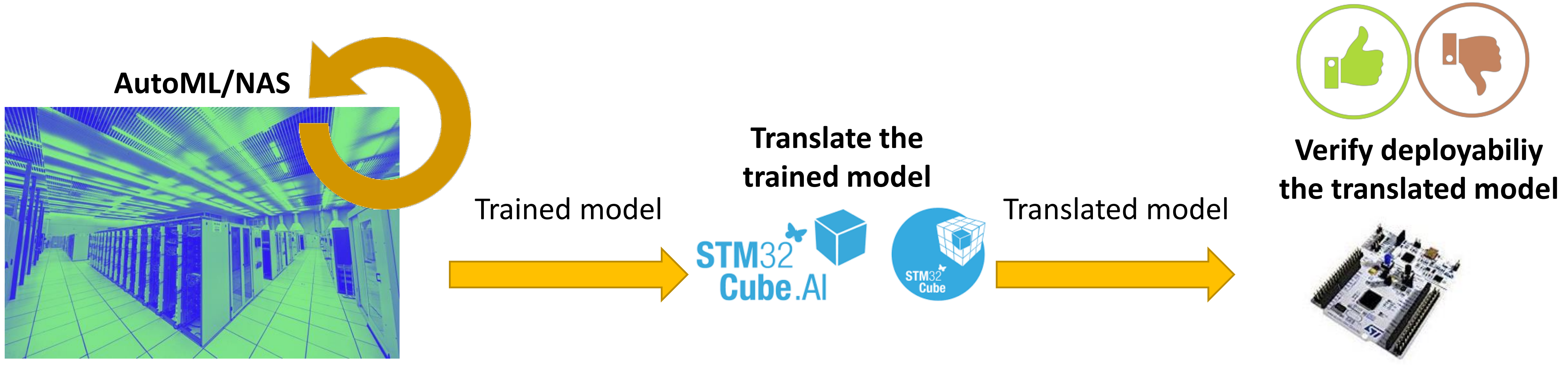
- AutoKeras
- Keras Tuner – scalable hyperparameter optimization framework: it comes with BO, Hyperband and Random Search algorithms built-in
- …



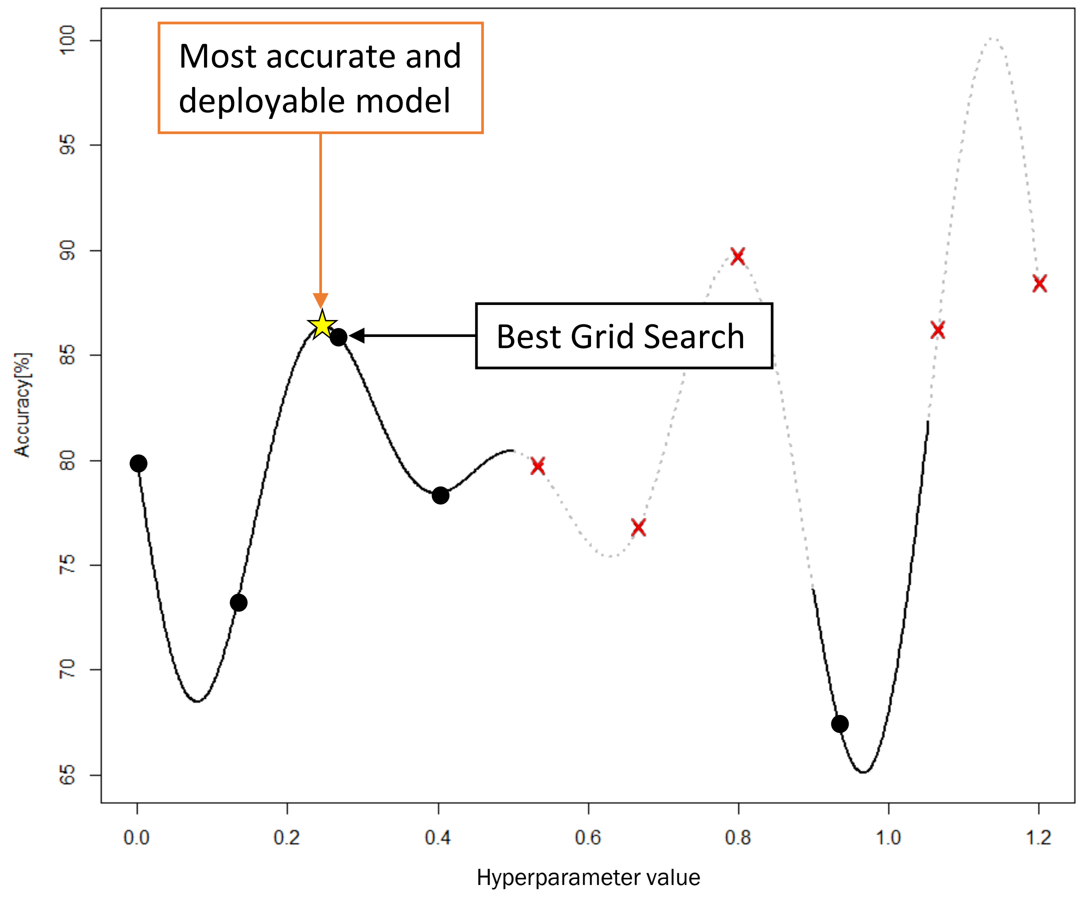
While many companies are currently leveraging on Cloud and specialized hardware (e.g., GPUs and TPUs) to train very accurate ML models, the need to deploy and run these models on tiny devices is emerging as the most relevant challenge, with a massive untapped market

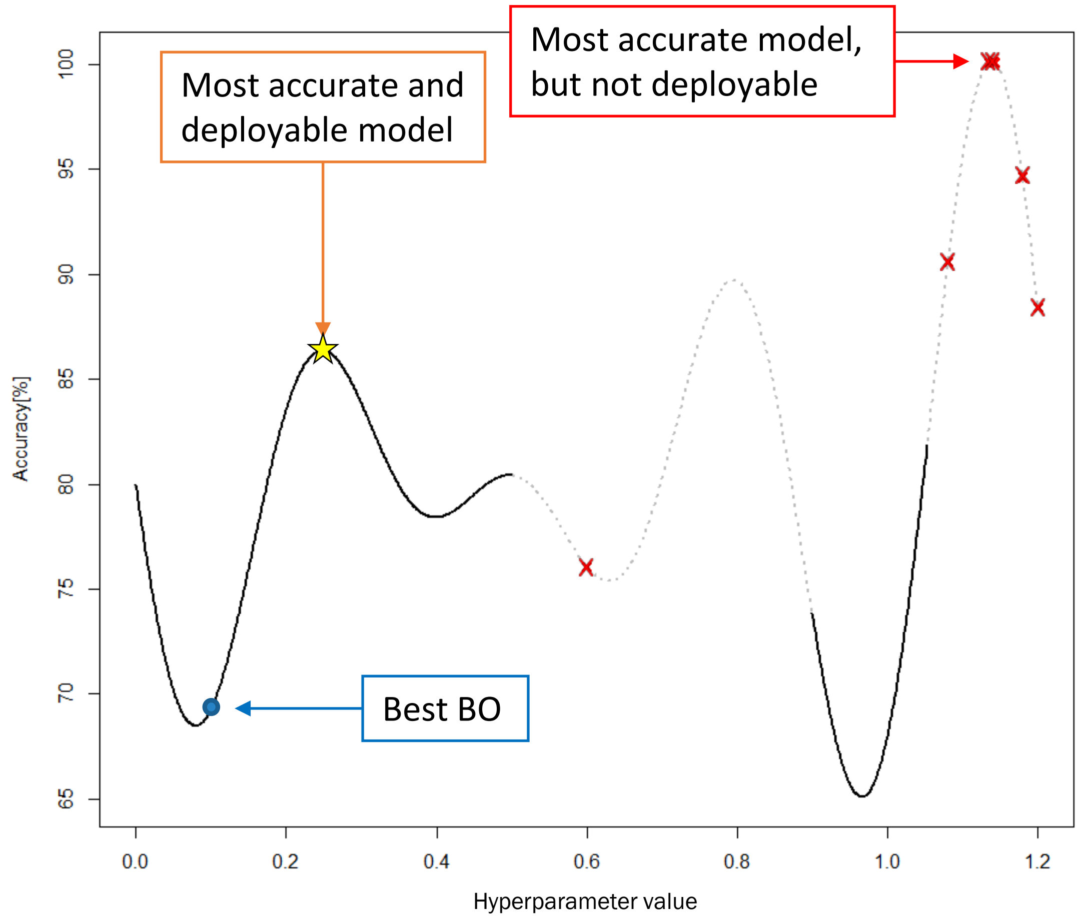
AUTOML/NAS ON "BIG" PLATFORMS

- AutoML and NAS frameworks can find accurate models within small number of trials, but they are typically performed on large computational platforms
- They cannot directly deal with *deployability,* leading to an accurate model which could result undeployable on a *tiny* device


AutoML/NAS
Trained model
Translate the trained model
STM32 Cube.AI
STM32 Cube
Translated model
Verify deployabiliy the translated model

LET'S GO BACK TO OUR EXAMPLE AND INCLUDE "*DEPLOYABILITY*"

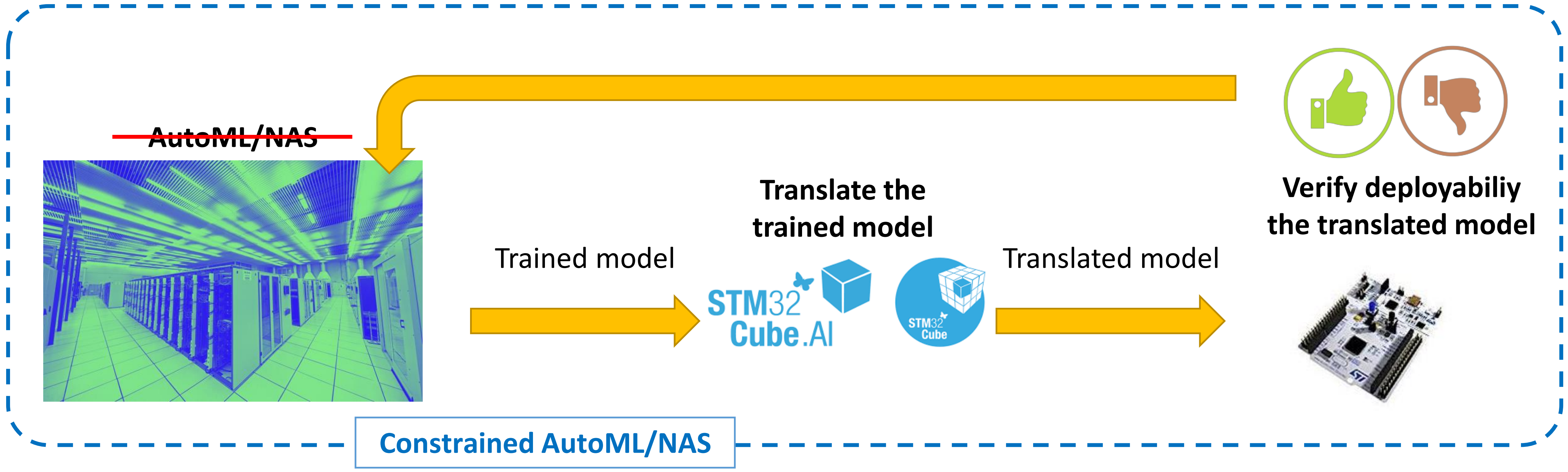




If BO does not include, into its loop, any information about deployability, the final result could be worse than Grid Search!

THE PROPOSED APPROACH: AUTOTINYML

Using information about "*deployability*" of each trained Neural Network to constrain BO, in order to identify both accurate and deployable models



AUTOTINYML

- We do not use multi-objective optimization because:
  - more complicated and computational expensive
  - solutions identified by multi-objective optimization are ***"trade-offs"*** between different goals (while we are interested in obtaining the most accurate model given the hardware resources of the tiny device)

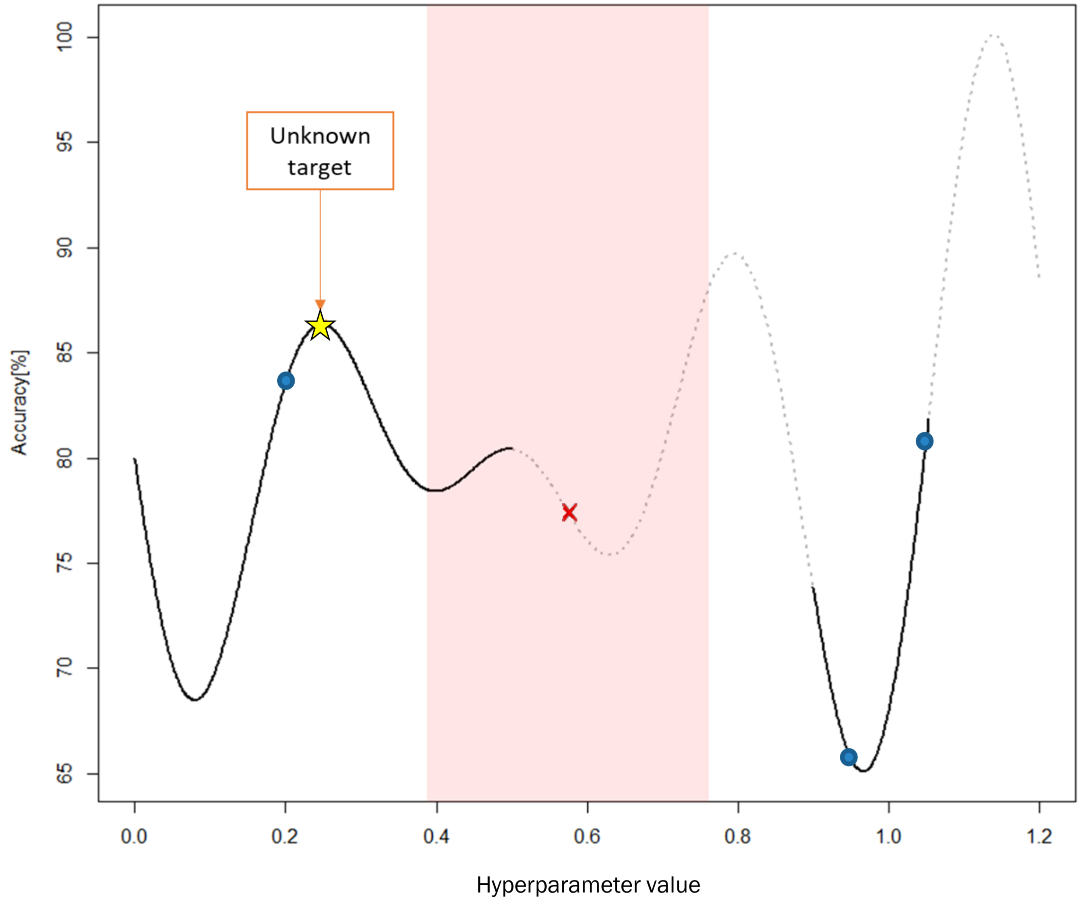
- We implemented a **"constrained"** AutoML/NAS framework, where:
  - also deployability is black-box as well as the objective function (e.g., Accuracy on *k* fold-cross validation)
  - **Phase 1** of the approach is **"deployability determination"**: hyperparameter configurations are sequentially selected to approximate the sub-region of the search space associated to deployable models
  - **Phase 2** of the approach is **"constrained BO"**: we perform BO only on the sub-region estimated to contain configurations of deployable models

LET'S USE AGAIN OUR EXAMPLE

PHASE 1 – Deployability Determination

- We start with 4 configurations randomly selected
- Compute their *k*FCV Accuracies and verify deployability
- 3 out of 4 are deployable, 1 not

- We can compute our first estimate of the "deployable" region (i.e., white region) by using a maximum margin separation classifier

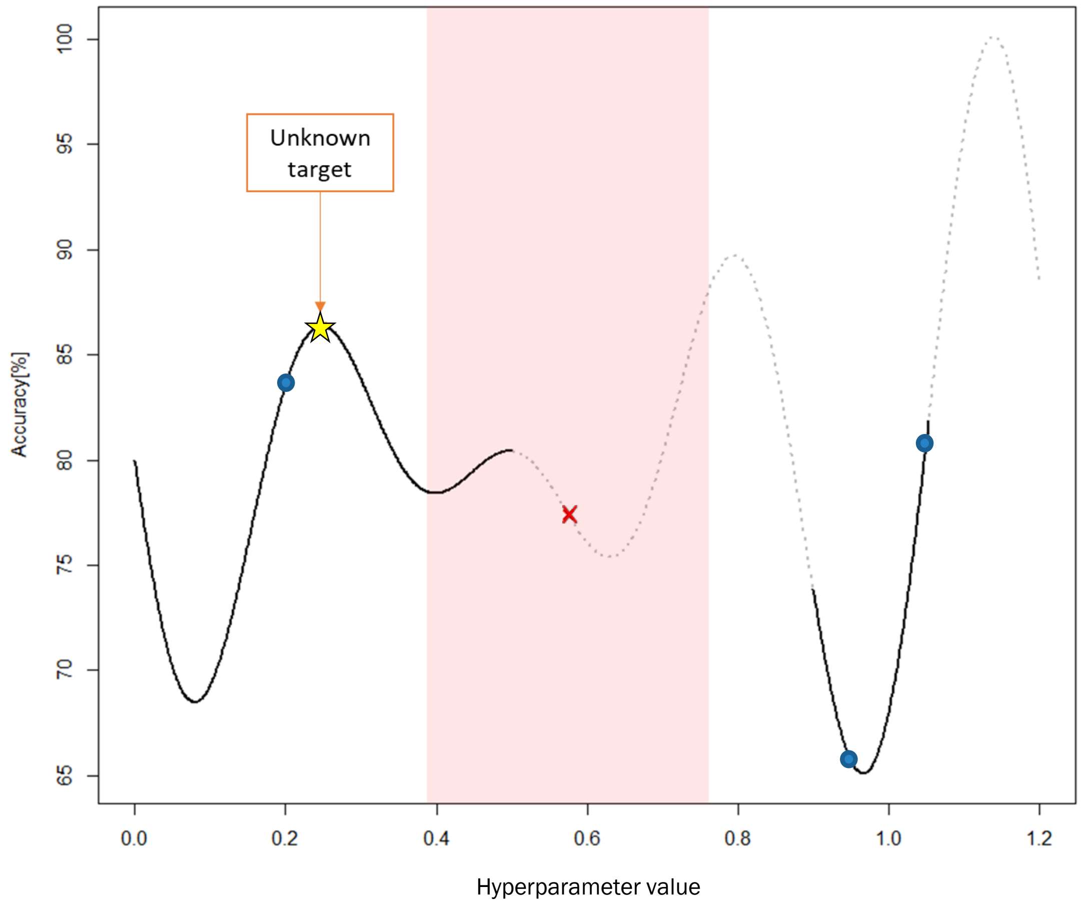


USE AGAIN OUR EXAMPLE

PHASE 1 – Deployability Determination

- The next configuration to evaluate is aimed (in this phase) at improving the approximation of the actual deployabile region

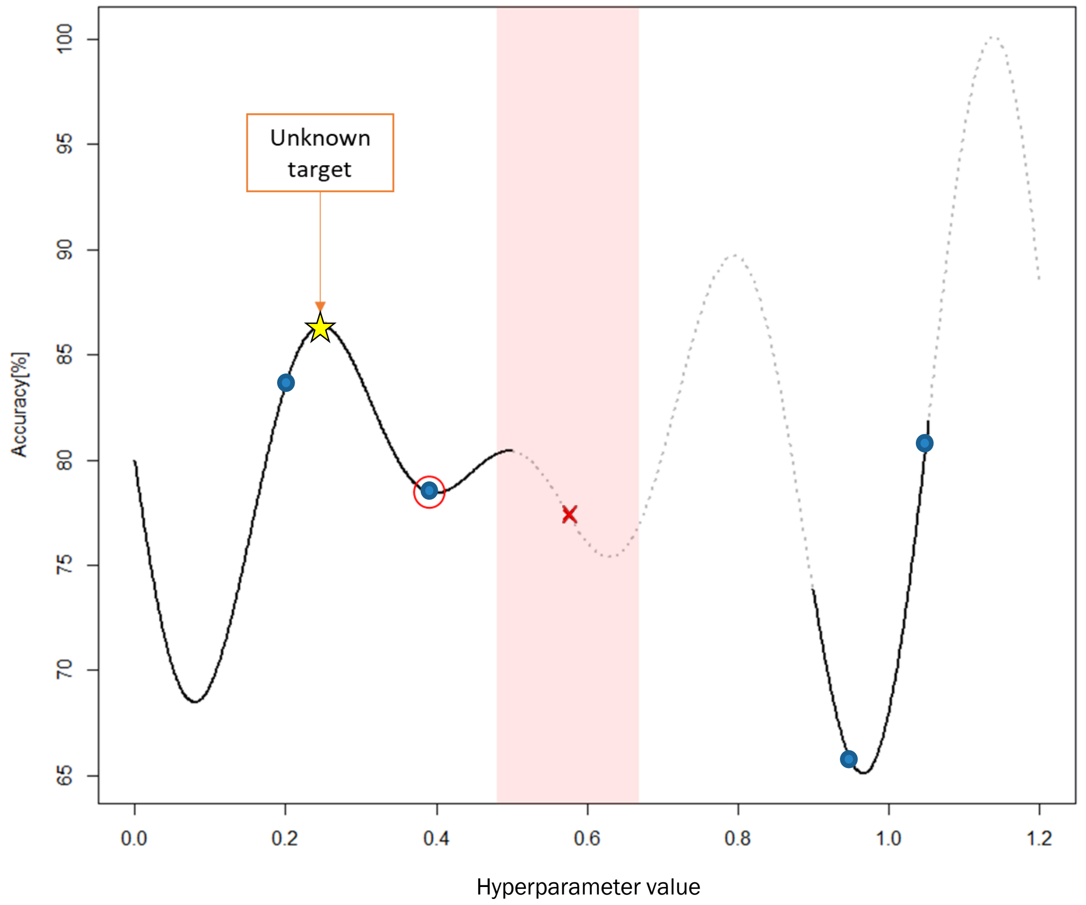
- So we select a configuration that is:
  - close to the separation boundaries
  - far from other configurations (to cover all the search space)



USE AGAIN OUR EXAMPLE

<u>PHASE 1 – Deployability Determination</u>

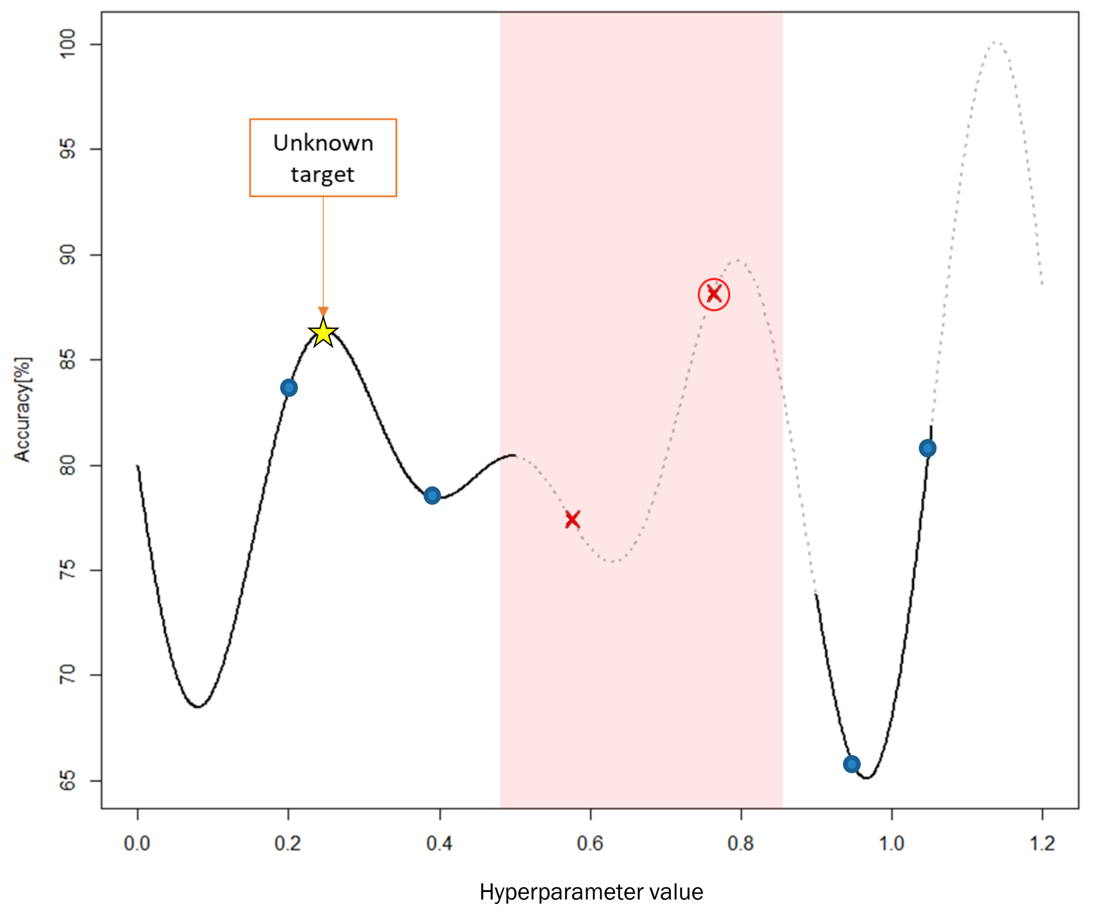
- Consequently, we update our estimate of the deployable region
- At this iteration we have a restriction of our initial estimate…
- Then, we select the next configuration…



USE AGAIN OUR EXAMPLE

PHASE 1 – Deployability Determination

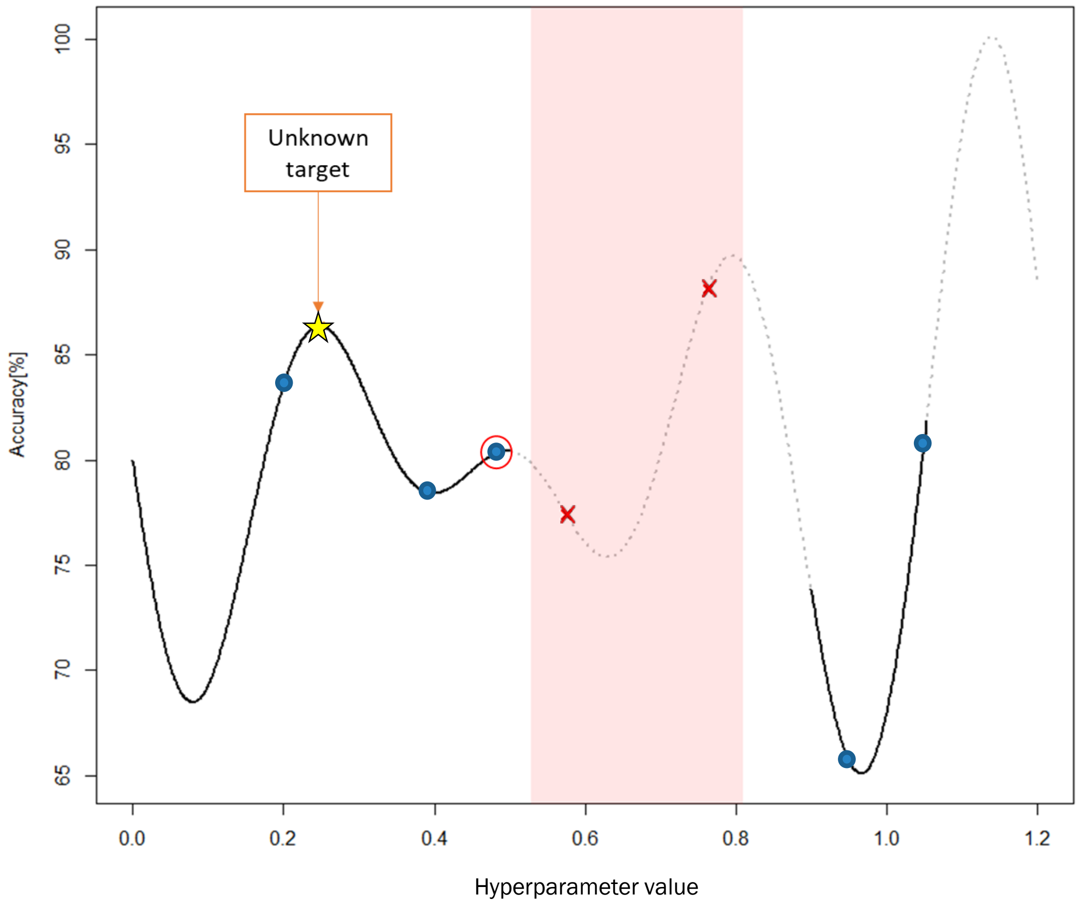
- At this iteration we have an expansion of our estimate…
- … we continue with the next configuration



USE AGAIN OUR EXAMPLE

PHASE 1 – Deployability Determination

- At this iteration we have a restriction of our estimate…
- Basically the estimate moves by *shrinking* or *expanding* its boundaries depending on the deployability of the evaluated configurations
- … we continue with the next configuration

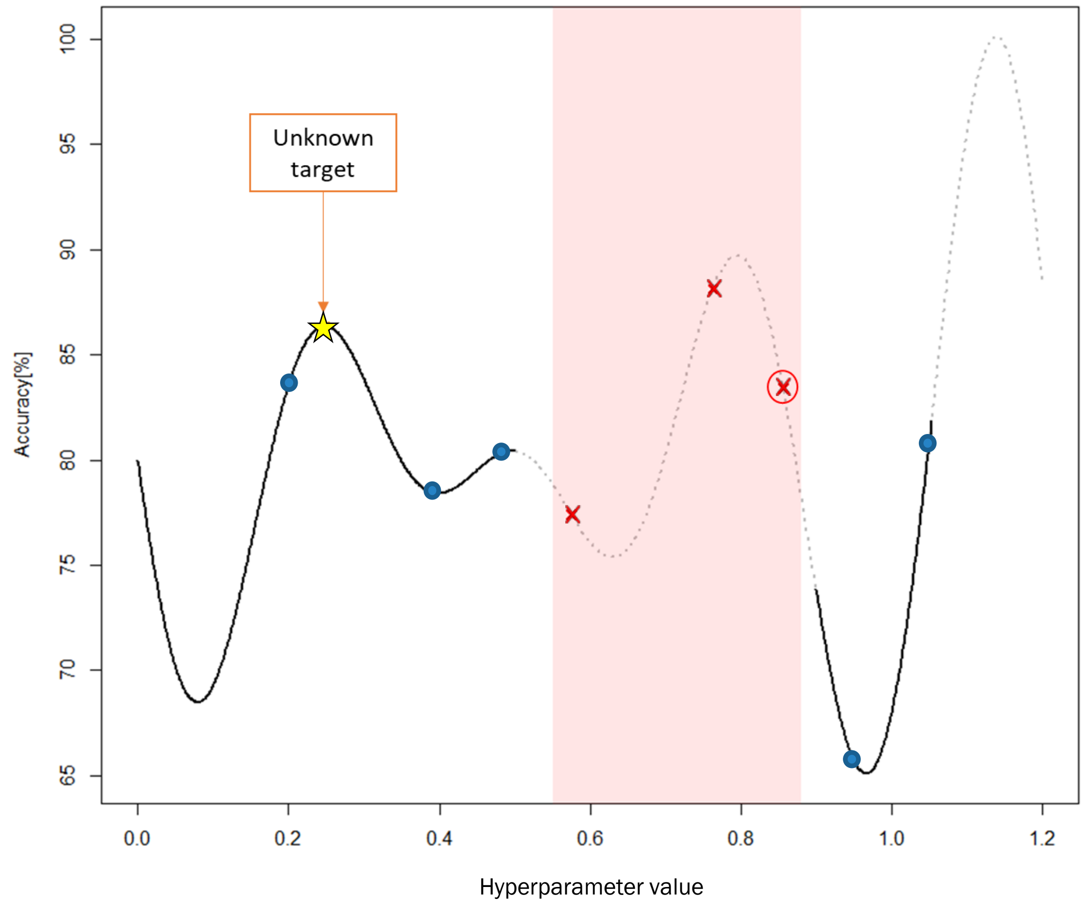


USE AGAIN OUR EXAMPLE

<u>PHASE 1 – Deployability Determination</u>

- At this iteration we have an expansion of the estimated deployability region

- … we can now move to PHASE 2!



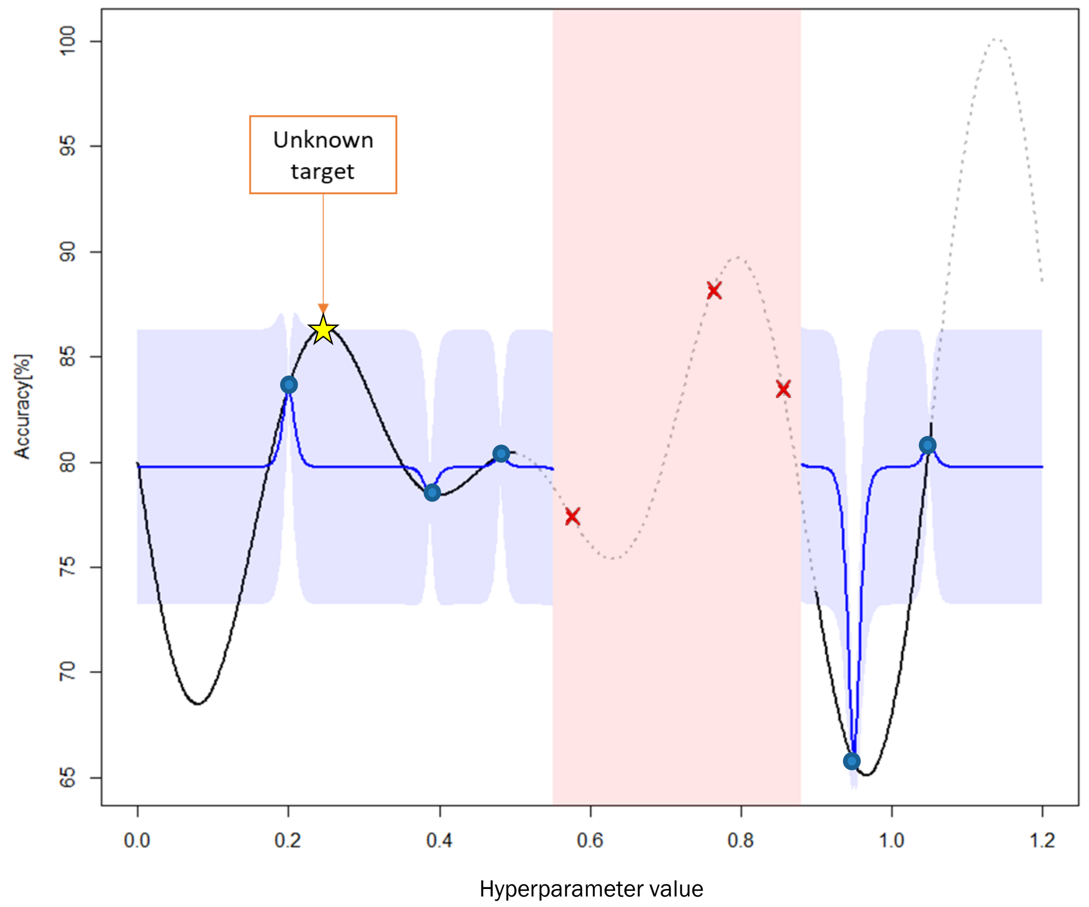
USE AGAIN OUR EXAMPLE

<u>PHASE 2 – Constrained BO</u>

- Even if the approximation of the depolyable region is not accurate, it is not an issue: the region is updated anytime a not-deployable configuration is selected

- We generate the probabilistic regression model only within the estimated deployability region

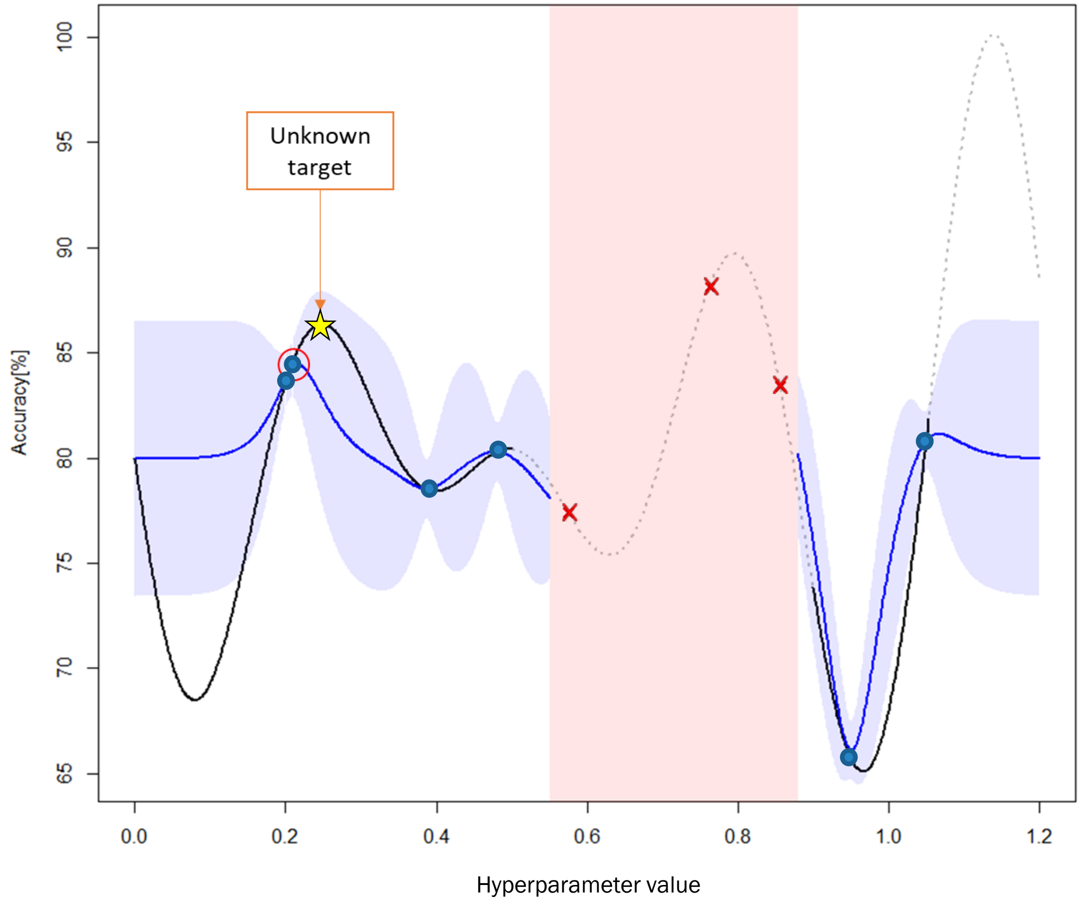
- Then select the next configuration according to Upper Confidence Bound



USE AGAIN OUR EXAMPLE

PHASE 2 – Constrained BO

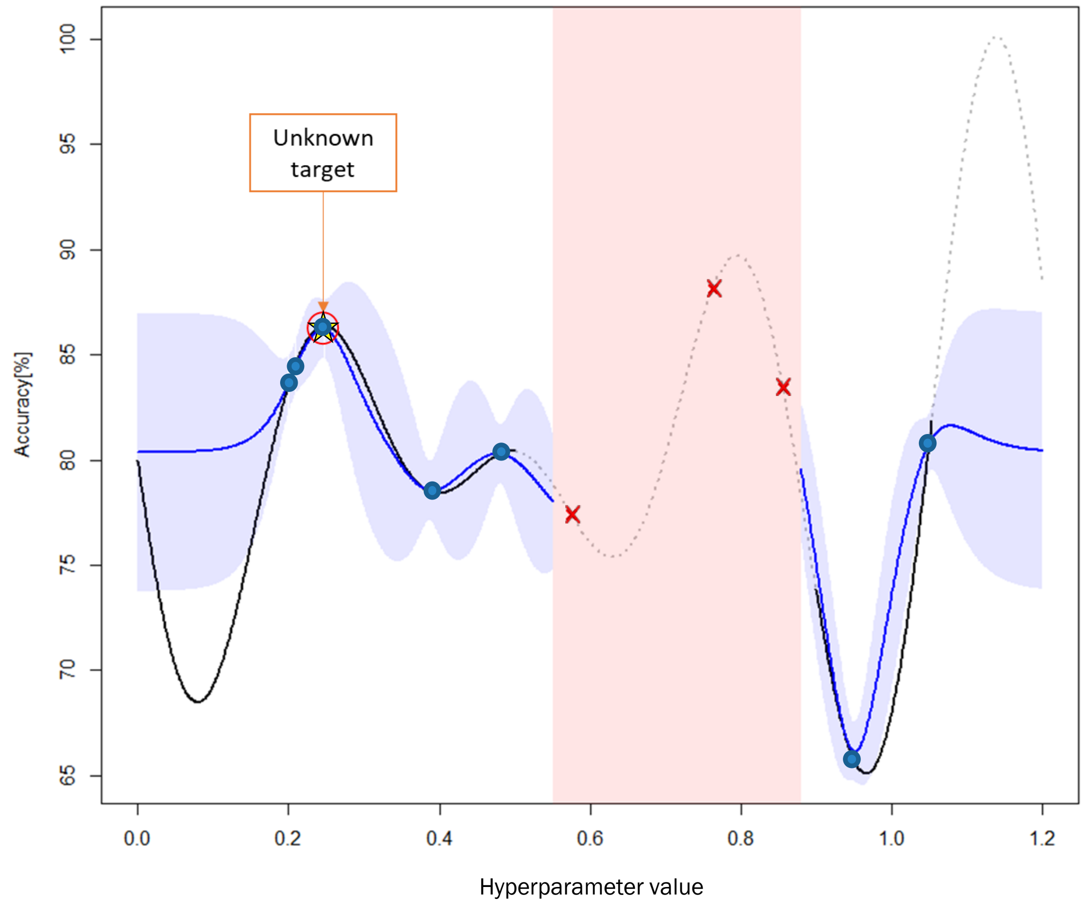
- The probabilistic regression model is updated
- And the next configuration to evaluate is selected…



USE AGAIN OUR EXAMPLE

<u>PHASE 2 – Constrained BO</u>

- We found the most accurate and deployable ML model!
- As in the previous cases (Grid Search and BO) we have used 10 trials! (4 initial, 4 Phase 1, 2 Phase 2)



Candelieri, A. (2019). Sequential model based optimization of partially defined functions under unknown constraints. *Journal of Global Optimization*, 1-23.

EXPERIMENT #1

A benchmark classification task:

- *User Identification From Walking Activity Data Set* (from the UCI Repository)

A baseline Neural Net:

- Convolutional Neural Network (CNN) for Human Activity Recognition (HAR), available on GitHub

Metrics to optimize:

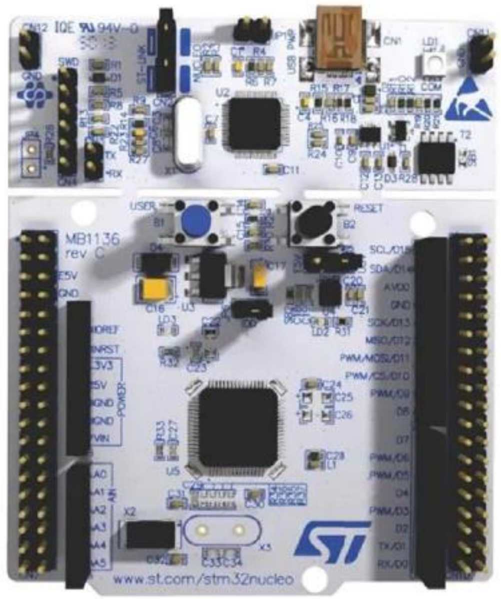
- Accuracy on a validation set

2 MCUs:

- STM32L476RGT6
- STM32F303K8T6

Deployability constraints:

- RAM≤128 KB (Big Board) / RAM ≤16 KB (Tiny Board)
- ROM≤1990 KB (Big Board) / ROM≤60 KB (Tiny Board)
- X-CROSS Accuracy 95% for both MCUs



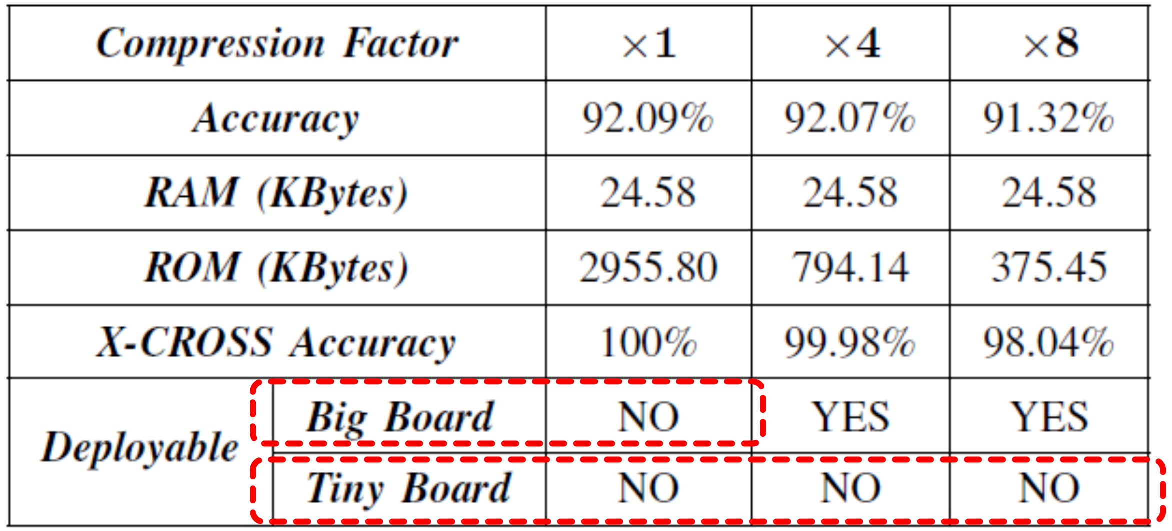
STM32L476RGT6

STM32F303K8T6

Hyperparameter		*Values*	
		Big Board	*Tiny Board*
Convolutional 2D	*Filters*	[16, 32, 64, 128, 256]	[8, 16, 32, 64, 128, 256]
	Activation	[ReLU, Sigmoid, Selu, Tanh]	
	Kernel Size	[1, 2]	
MaxPooling	*Windows Pooling*	[1, 2]	
Dropout		[0.1, 0.2, 0.3]	
Dense	*Units*	[16, 32, 64, 128, 256]	[8, 16, 32, 64, 128, 256]
	Activation	[ReLU, Sigmoid, Selu, Tanh]	
Optimizer		[Adam, SGD, RMSProp, Adadelta, Adamax]	

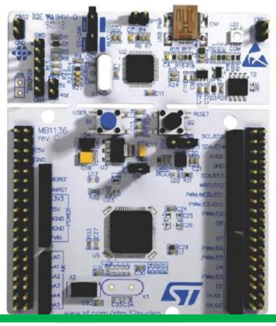
DEPLOYABILITY OF THE BASELINE CNN

- The CNN provided on GitHub (namely "Baseline") resulted to be deployable only on the "Big Board" with a compression factor x4 and x8
- We used our AutoTinyML to optimize the CNN's hyperparameters while keeping fixed its architecture



Compression Factor		$\times 1$	$\times 4$	$\times 8$
Accuracy		92.09%	92.07%	91.32%
RAM (KBytes)		24.58	24.58	24.58
ROM (KBytes)		2955.80	794.14	375.45
X-CROSS Accuracy		100%	99.98%	98.04%
Deployable	*Big Board*	NO	YES	YES
	Tiny Board	NO	NO	NO

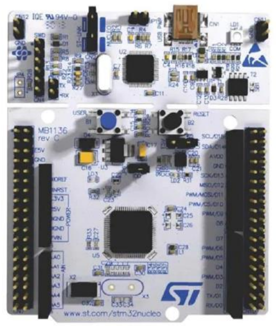
AUTOTINYML RESULTS



	Compression ×1		*Compression ×4*	
	Baseline	***AutoTinyML***	***Baseline***	***AutoTinyML***
Accuracy	92.09%	92.44% (+/- 0.64)	92.07%	92.93% (+/- 0.55)
RAM	24.57	9.75 (+/- 5.11)	24.57	15.16 (+/- 7.92)
ROM	2955.80	688.30 (+/- 144.42)	794.13	546.31 (+/- 283.34)
X-CROSS	100%	100% (+/- 0.00)	99.98%	99.70% (+/- 0.53)
MACC	874970	269327 (+/- 83895)	874970	784767.33 (+/- 318078)

	Compression ×4		*Compression ×8*	
	Baseline	***AutoTinyML***	***Baseline***	***AutoTinyML***
Accuracy	92.07%	88.27% (+/- 2.47)	91.32%	91.41% (+/- 0.85)
RAM	24.57	5.17 (+/- 3.04)	24.57	5.42 (+/- 3.22)
ROM	794.13	44.93 (+/- 12.34)	375.45	49.51 (+/- 11.04)
X-CROSS	99.98%	99.86% (+/- 0.03)	98.04%	98.42% (+/- 0.59)
MACC	874970	87110 (+/- 34747)	874970	159714 (+/- 61613)

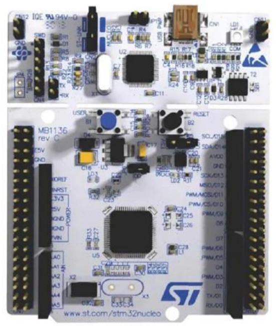
RESULTS



	Compression ×1		Compression ×4	
	Baseline	*AutoTinyML*	*Baseline*	*AutoTinyML*
Accuracy	92.09%	92.44% (+/- 0.64)	92.07%	92.93% (+/- 0.55)
RAM	24.57	9.75 (+/- 5.11)	24.57	15.16 (+/- 7.92)
ROM	2955.80	688.30 (+/- 144.42)	794.13	546.31 (+/- 283.34)
X-CROSS	100%	100% (+/- 0.00)	99.98%	99.70% (+/- 0.53)
MACC	874970	269327 (+/- 83895)	874970	784767.33 (+/- 318078)

	Compression ×4		Compression ×8	
	Baseline	*AutoTinyML*	*Baseline*	*AutoTinyML*
Accuracy	92.07%	88.27% (+/- 2.47)	91.32%	91.41% (+/- 0.85)
RAM	24.57	5.17 (+/- 3.04)	24.57	5.42 (+/- 3.22)
ROM	794.13	44.93 (+/- 12.34)	375.45	49.51 (+/- 11.04)
X-CROSS	99.98%	99.86% (+/- 0.03)	98.04%	98.42% (+/- 0.59)
MACC	874970	87110 (+/- 34747)	874970	159714 (+/- 61613)

RESULTS

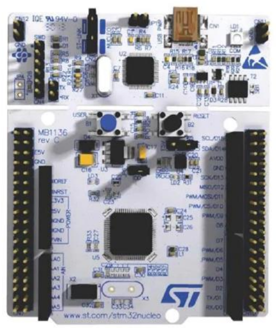


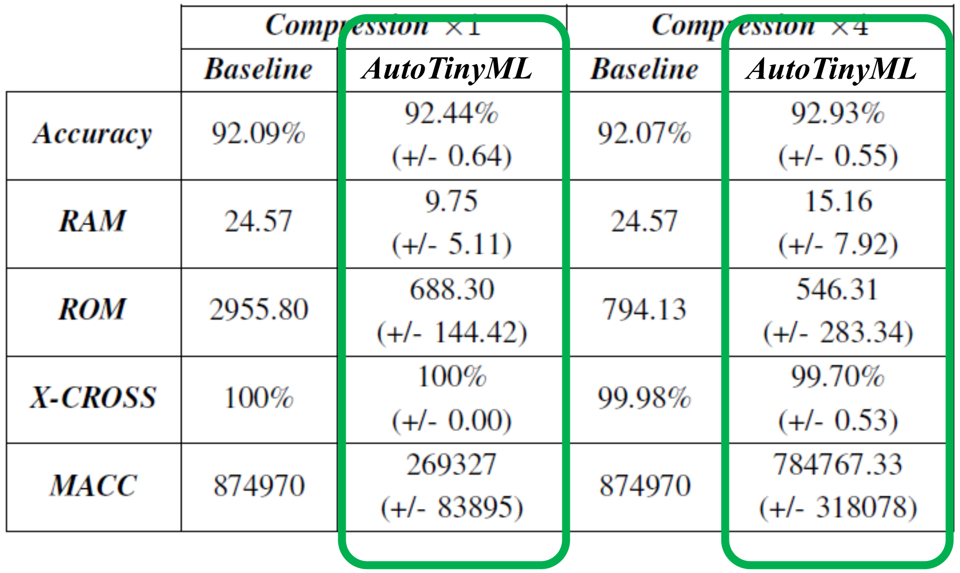
	Compression ×1		*Compression ×4*	
	Baseline	***AutoTinyML***	***Baseline***	***AutoTinyML***
Accuracy	92.09%	92.44% (+/- 0.64)	92.07%	92.93% (+/- 0.55)
RAM	24.57	9.75 (+/- 5.11)	24.57	15.16 (+/- 7.92)
ROM	2955.80	688.30 (+/- 144.42)	794.13	546.31 (+/- 283.34)
X-CROSS	100%	100% (+/- 0.00)	99.98%	99.70% (+/- 0.53)
MACC	874970	269327 (+/- 83895)	874970	784767.33 (+/- 318078)

	Compression ×4		*Compression ×8*	
	Baseline	***AutoTinyML***	***Baseline***	***AutoTinyML***
Accuracy	92.07%	88.27% (+/- 2.47)	91.32%	91.41% (+/- 0.85)
RAM	24.57	5.17 (+/- 3.04)	24.57	5.42 (+/- 3.22)
ROM	794.13	44.93 (+/- 12.34)	375.45	49.51 (+/- 11.04)
X-CROSS	99.98%	99.86% (+/- 0.03)	98.04%	98.42% (+/- 0.59)
MACC	874970	87110 (+/- 34747)	874970	159714 (+/- 61613)

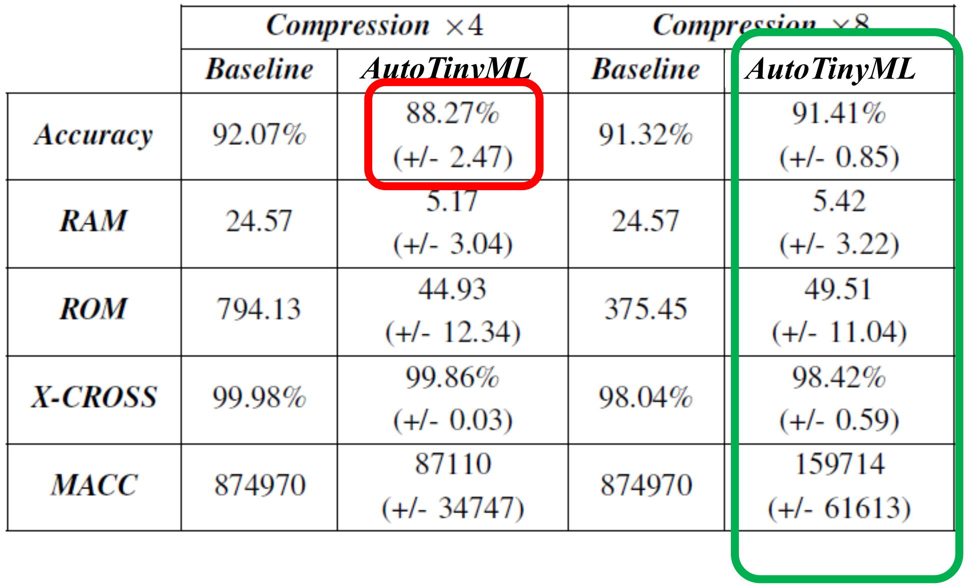
RESULTS

Perego, R., Candelieri, A., Archetti, F., & Pau, D. (2020, September). *Tuning deep neural network's hyperparameters constrained to deployability on tiny systems*. In International Conference on Artificial Neural Networks (pp. 92-103). Springer, Cham.





	Compression ×1		*Compression ×4*	
	Baseline	***AutoTinyML***	***Baseline***	***AutoTinyML***
Accuracy	92.09%	92.44% (+/- 0.64)	92.07%	92.93% (+/- 0.55)
RAM	24.57	9.75 (+/- 5.11)	24.57	15.16 (+/- 7.92)
ROM	2955.80	688.30 (+/- 144.42)	794.13	546.31 (+/- 283.34)
X-CROSS	100%	100% (+/- 0.00)	99.98%	99.70% (+/- 0.53)
MACC	874970	269327 (+/- 83895)	874970	784767.33 (+/- 318078)



	Compression ×4		*Compression ×8*	
	Baseline	***AutoTinyML***	***Baseline***	***AutoTinyML***
Accuracy	92.07%	88.27% (+/- 2.47)	91.32%	91.41% (+/- 0.85)
RAM	24.57	5.17 (+/- 3.04)	24.57	5.42 (+/- 3.22)
ROM	794.13	44.93 (+/- 12.34)	375.45	49.51 (+/- 11.04)
X-CROSS	99.98%	99.86% (+/- 0.03)	98.04%	98.42% (+/- 0.59)
MACC	874970	87110 (+/- 34747)	874970	159714 (+/- 61613)

EXPERIMENT #2

A benchmark classification task:

- *Bottle level classification*

A baseline Neural Net:

- Convolutional Neural Network (CNN) for Bottle level classification (manually tuned)

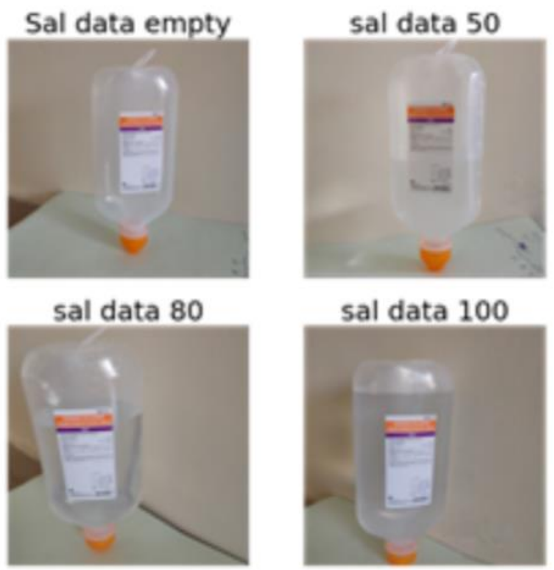
Metrics to optimize:

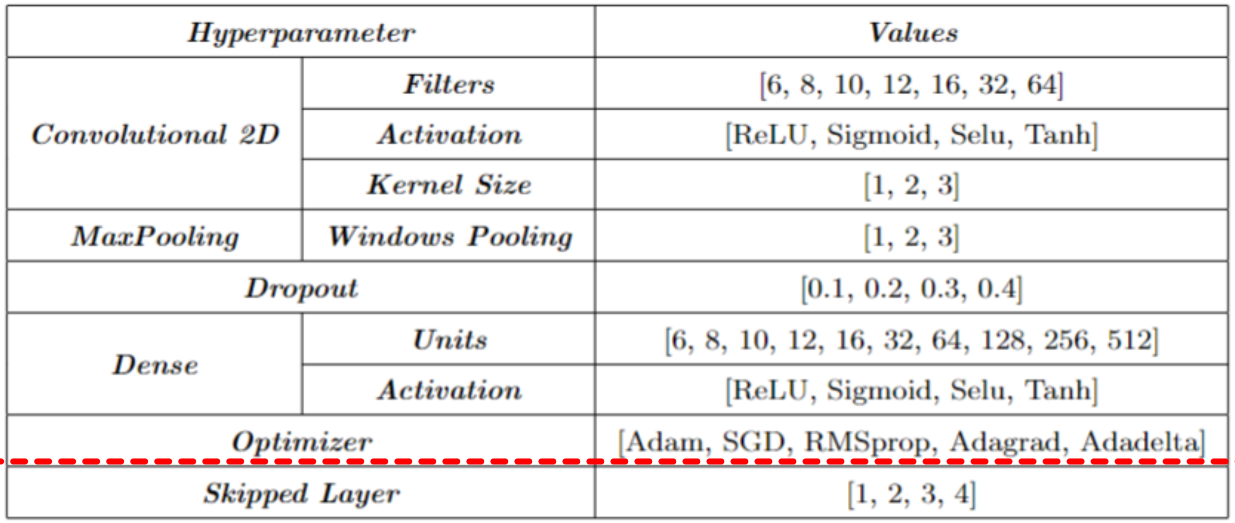
- Accuracy on a validation set

Same MCUs and deployability constraints:

- STM32L476RGT6
- STM32F303K8T6

One architectural hyperparameter included

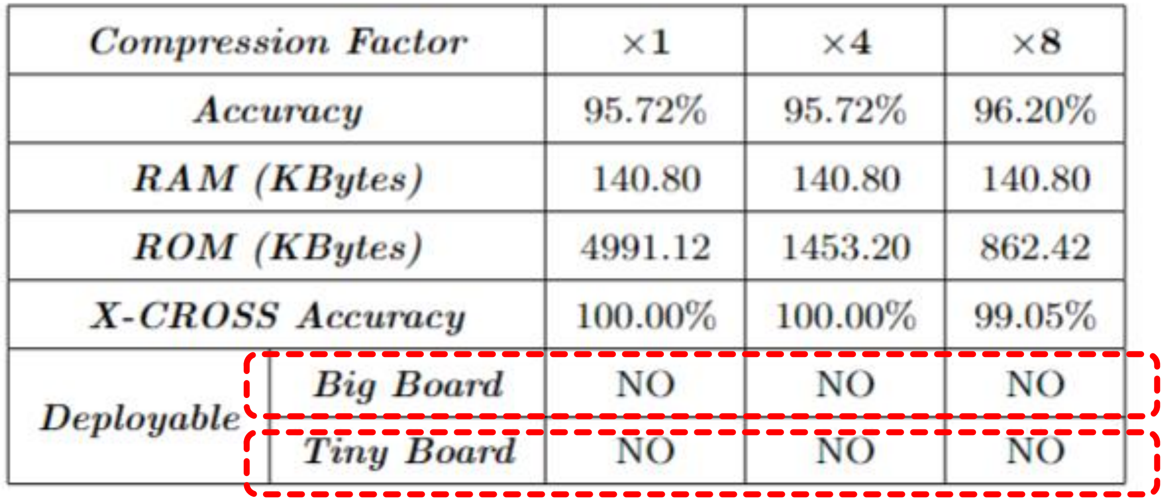




Hyperparameter		*Values*
Convolutional 2D	*Filters*	[6, 8, 10, 12, 16, 32, 64]
	Activation	[ReLU, Sigmoid, Selu, Tanh]
	Kernel Size	[1, 2, 3]
MaxPooling	*Windows Pooling*	[1, 2, 3]
Dropout		[0.1, 0.2, 0.3, 0.4]
Dense	*Units*	[6, 8, 10, 12, 16, 32, 64, 128, 256, 512]
	Activation	[ReLU, Sigmoid, Selu, Tanh]
Optimizer		[Adam, SGD, RMSprop, Adagrad, Adadelta]
Skipped Layer		[1, 2, 3, 4]

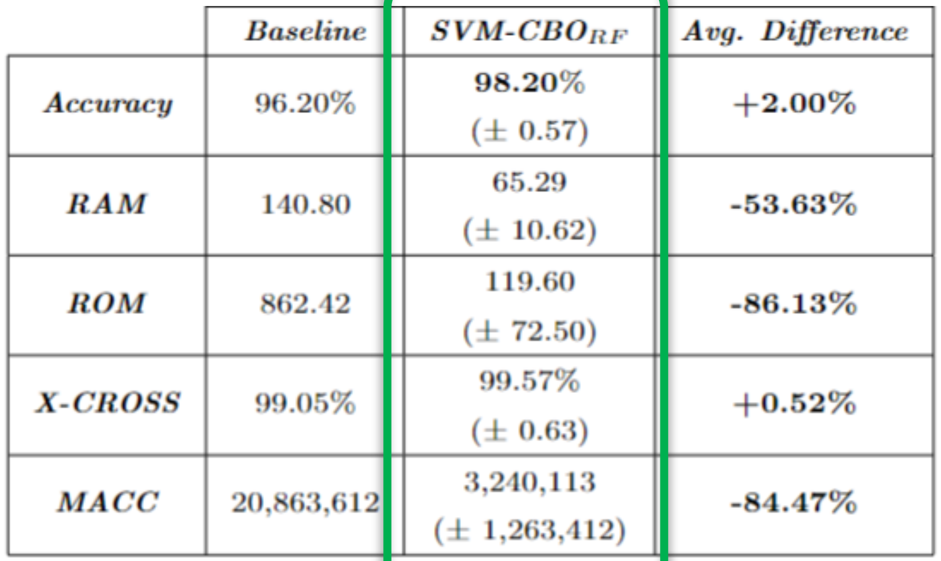
DEPLOYABILITY OF THE BASELINE CNN

- The baseline CNN is not deployable on the two boards!

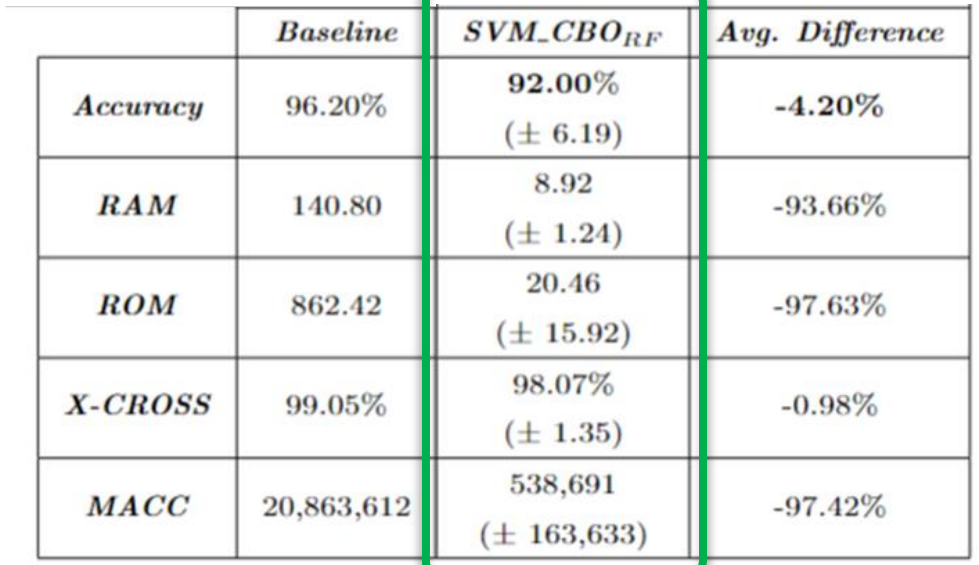


Compression Factor		$\times 1$	$\times 4$	$\times 8$
Accuracy		95.72%	95.72%	96.20%
RAM (KBytes)		140.80	140.80	140.80
ROM (KBytes)		4991.12	1453.20	862.42
X-CROSS Accuracy		100.00%	100.00%	99.05%
Deployable	*Big Board*	NO	NO	NO
	Tiny Board	NO	NO	NO

RESULTS



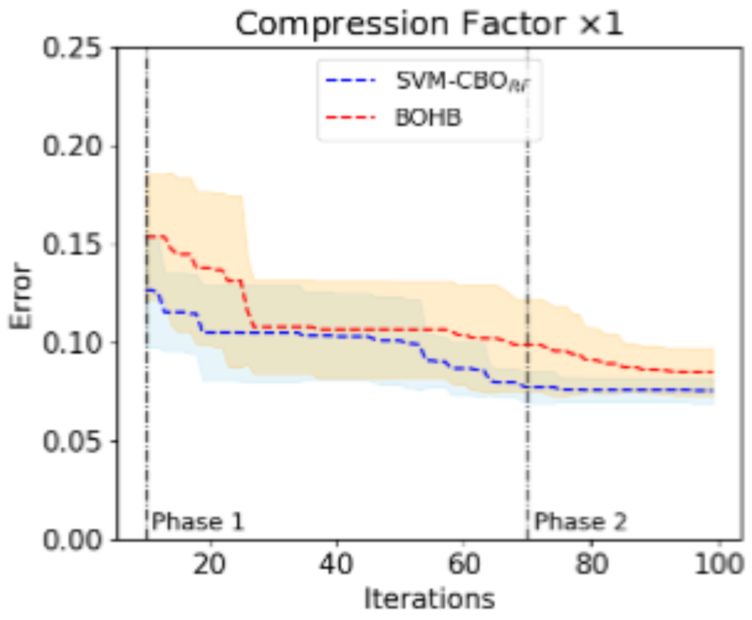
	Baseline	*SVM-CBO$_{RF}$*	*Avg. Difference*
Accuracy	96.20%	**98.20%** (± 0.57)	**+2.00%**
RAM	140.80	65.29 (± 10.62)	**-53.63%**
ROM	862.42	119.60 (± 72.50)	**-86.13%**
X-CROSS	99.05%	99.57% (± 0.63)	**+0.52%**
MACC	20,863,612	3,240,113 (± 1,263,412)	**-84.47%**

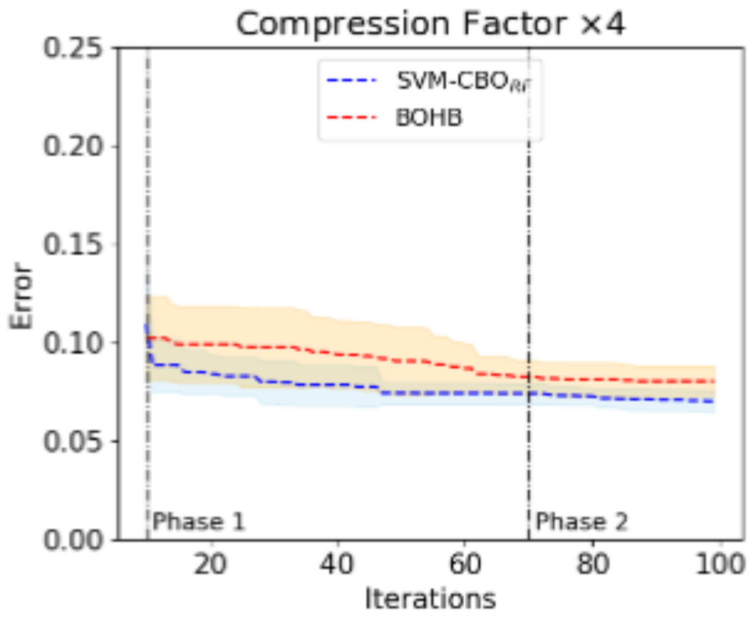


	Baseline	*SVM_CBO$_{RF}$*	*Avg. Difference*
Accuracy	96.20%	**92.00%** (± 6.19)	**-4.20%**
RAM	140.80	8.92 (± 1.24)	-93.66%
ROM	862.42	20.46 (± 15.92)	-97.63%
X-CROSS	99.05%	98.07% (± 1.35)	-0.98%
MACC	20,863,612	538,691 (± 163,633)	-97.42%

COMPARISON AGAINST OTHER AUTOML TOOLS (I.E., BOHB)

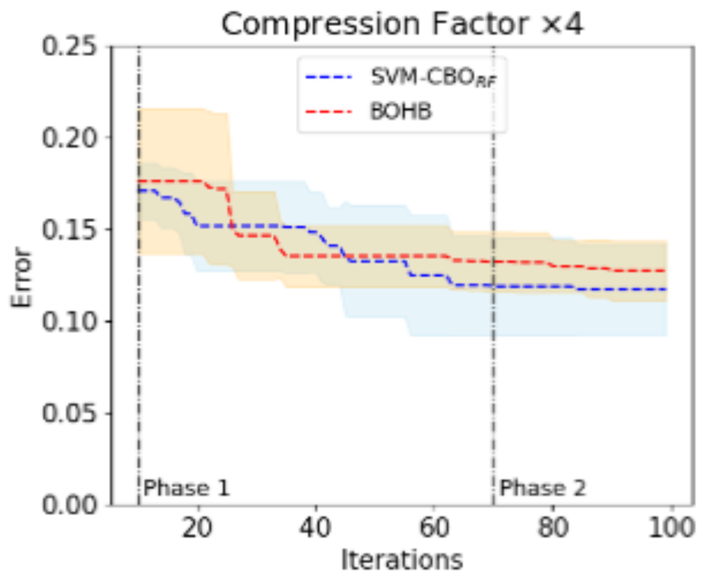
Misclassification Error on kFCV (100%-Accuracy)

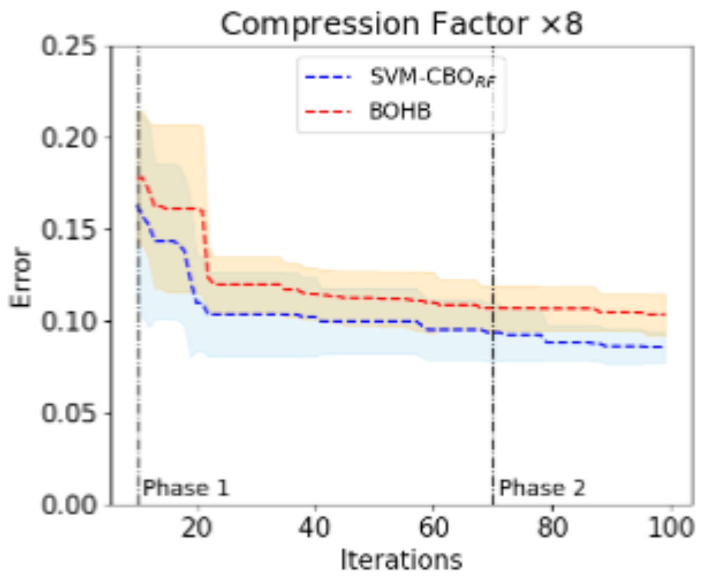


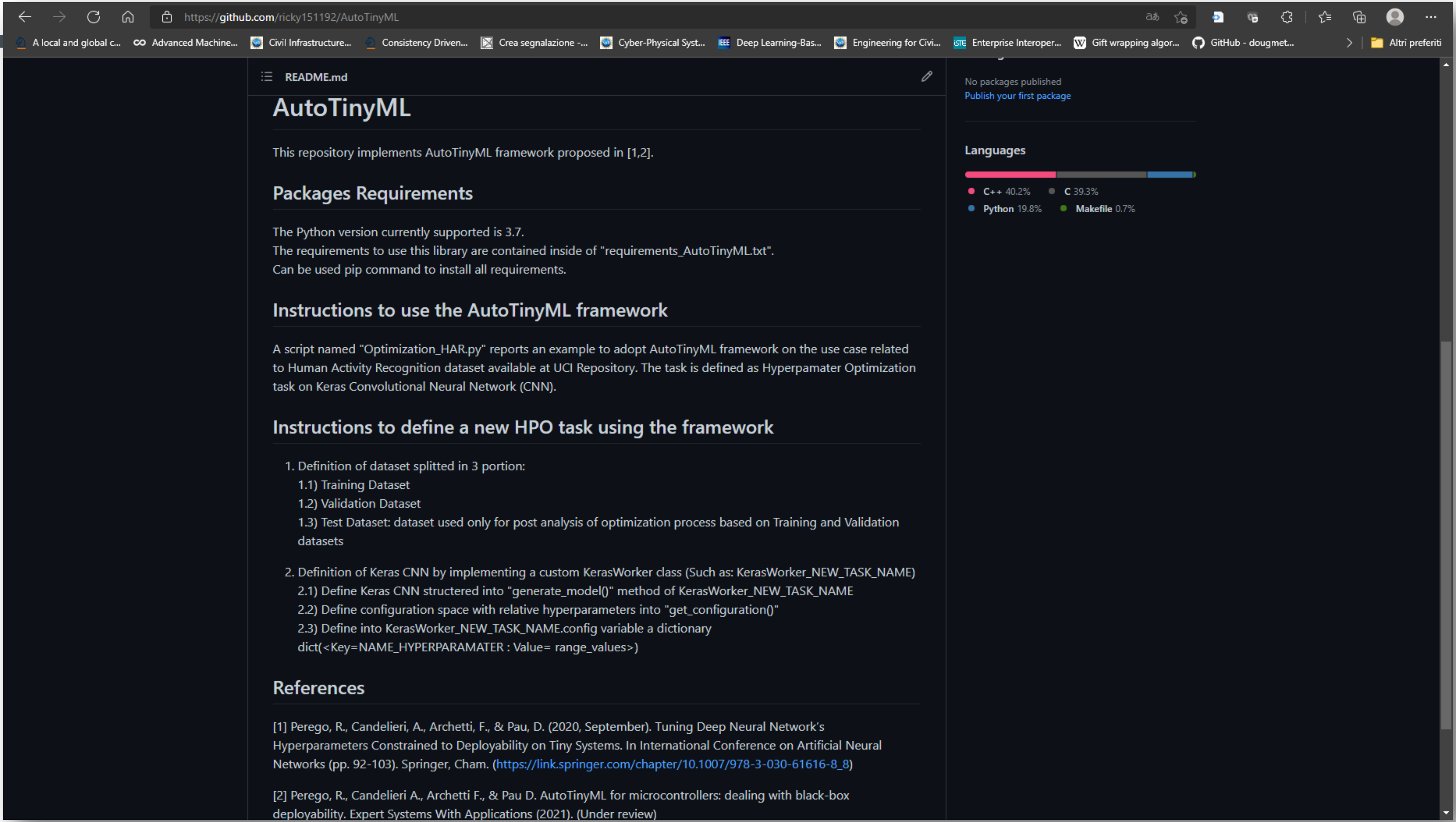


COMPARISON AGAINST OTHER AUTOML TOOLS (I.E., BOHB)

Misclassification Error on kFCV (100%-Accuracy)






https://github.com/ricky151192/AutoTinyML

AutoTinyML

This repository implements AutoTinyML framework proposed in [1,2].

Packages Requirements

The Python version currently supported is 3.7.
The requirements to use this library are contained inside of "requirements_AutoTinyML.txt".
Can be used pip command to install all requirements.

Instructions to use the AutoTinyML framework

A script named "Optimization_HAR.py" reports an example to adopt AutoTinyML framework on the use case related to Human Activity Recognition dataset available at UCI Repository. The task is defined as Hyperpamater Optimization task on Keras Convolutional Neural Network (CNN).

Instructions to define a new HPO task using the framework

1. Definition of dataset splitted in 3 portion:
   1.1) Training Dataset
   1.2) Validation Dataset
   1.3) Test Dataset: dataset used only for post analysis of optimization process based on Training and Validation datasets
2. Definition of Keras CNN by implementing a custom KerasWorker class (Such as: KerasWorker_NEW_TASK_NAME)
   2.1) Define Keras CNN structered into "generate_model()" method of KerasWorker_NEW_TASK_NAME
   2.2) Define configuration space with relative hyperparameters into "get_configuration()"
   2.3) Define into KerasWorker_NEW_TASK_NAME.config variable a dictionary dict(<Key=NAME_HYPERPARAMATER : Value= range_values>)

References

[1] Perego, R., Candelieri, A., Archetti, F., & Pau, D. (2020, September). Tuning Deep Neural Network's Hyperparameters Constrained to Deployability on Tiny Systems. In International Conference on Artificial Neural Networks (pp. 92-103). Springer, Cham. (https://link.springer.com/chapter/10.1007/978-3-030-61616-8_8)

[2] Perego, R., Candelieri A., Archetti F., & Pau D. AutoTinyML for microcontrollers: dealing with black-box deployability. Expert Systems With Applications (2021). (Under review)

No packages published
Publish your first package

Languages

● C++ 40.2% ● C 39.3%
● Python 19.8% ● Makefile 0.7%

CONCLUSIONS

- The approach allows to obtain accurate and deployable models, also on very tiny devices (the smallest STM MCUs), without requiring any further model compression or pruning!

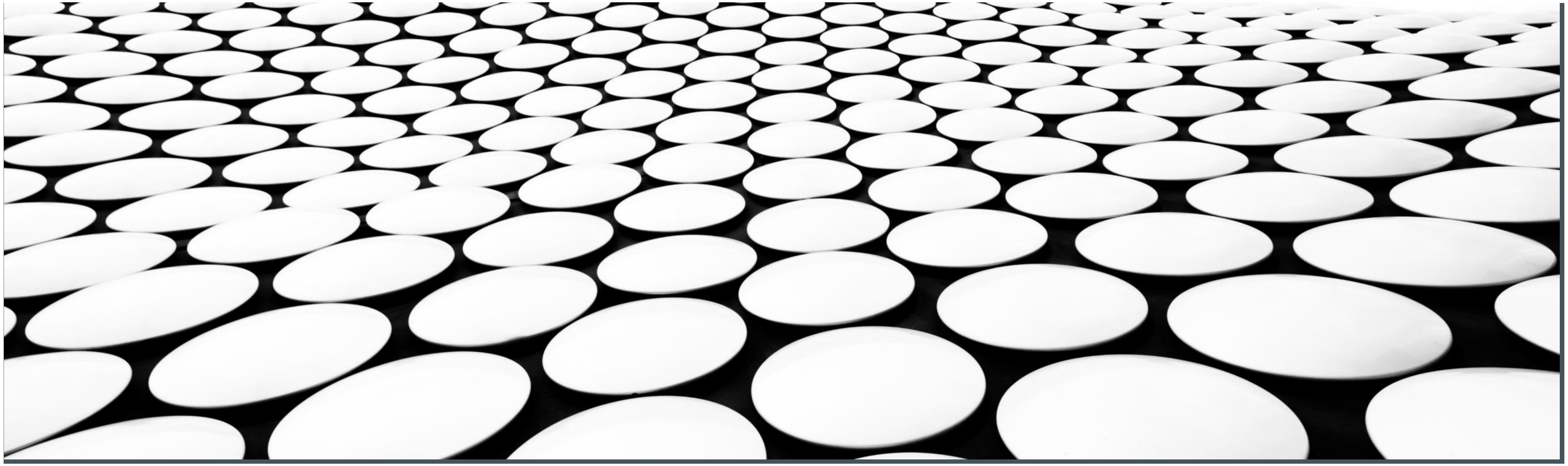
- In any case, model pruning or compression can be also applied in order to further reduce, if needed, the NN size or complexity

- Our tool exploits constraints related to MCU's hardware resources, differently from multi-objective strategies recently proposed for resource-efficient AutoML/NAS on large platforms:
- Indeed, we are interested in searching for the most accurate model given the hardware limitations of the tiny device, instead of searching for a trade-off between accuracy and resource-efficiency

- Including a further constraint (or an objective) related to MACC will allow us to address also requirements on *latency of the prediction* and *power/battery* management

THANK YOU!

antonio.candelieri@unimib.it



TINY ML SUMMIT

tinyML Summit 2022 Sponsors

ANALOG DEVICES AHEAD OF WHAT'S POSSIBLE™ · AONdevices · arm · ASPINITY · brainchip The Neuromorphic Computing Company · CEVA®

Deeplite · EDGE IMPULSE · emza visual sense · FotaHub · GREENWAVES TECHNOLOGIES · Grovety Inc.

Himax · HOTG · imagimob · infineon · itemis · KLIKA TECH GLOBAL IOT SOLUTIONS

LatentAI · LATTICE SEMICONDUCTOR · Micro.ai · OmniML · NXP · PX:I · Plumerai

PROPHESEE · Qeexo · Qualcomm · Rackner · RealityAI® Engineering Solutions for the Edge · REEXEN technology

RENESAS · SAP · seeed studio The IoT Hardware Enabler · SensiML™ · Sony Semiconductor Solutions Corporation · ST life.augmented · SA STREAM ANALYZE

synaptics® · SynSense · SYNTIANT · Tensil.ai · TensorFlow · XMOS

Copyright Notice