1st Workshop on Distributed Edge AI – Risks and Challenges (DE-AI’23)
18th Conference on Computer Science and Intelligence Systems (FedCSIS 2023)
Warsaw, Poland – 18 September 2023

Comparing Performance of Machine Learning Tools across Computing Platforms

Pedro Vicente[1,3], Pedro M. Santos[1,3], Barikisu Asulba[2,3],
Nuno Martins[4], Joana Sousa[4], Luís Almeida[2,3]

1 - Instituto Superior de Engenharia do Porto - Instituto Politécnico do Porto
1 - Universidade do Porto - Faculdade de Engenharia
3 - CISTER Research Unit
4 - NOS Inovação, Lisboa, Portugal

Project:

Funding:

Cofinanciado por:

Introduction

Context

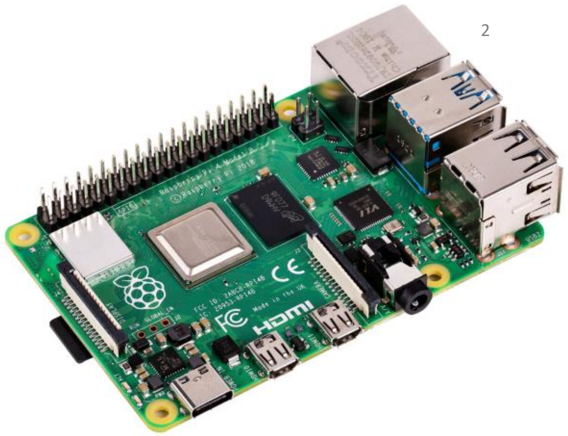
- Embedded systems (ES) are wide-spread in our world and responsible for many critical systems
- Machine learning (ML) tools have become a well-established solution for data-intensive tasks

Challenge

- Embedded systems are typically resource-constrained platforms, ranging from micro-controllers to small-scale x86-32 bit platforms (e.g., ARM)
- While there is a plethora of ML libraries, not all provide the small memory footprint and ability for stand-alone operation necessary for embedded systems

1

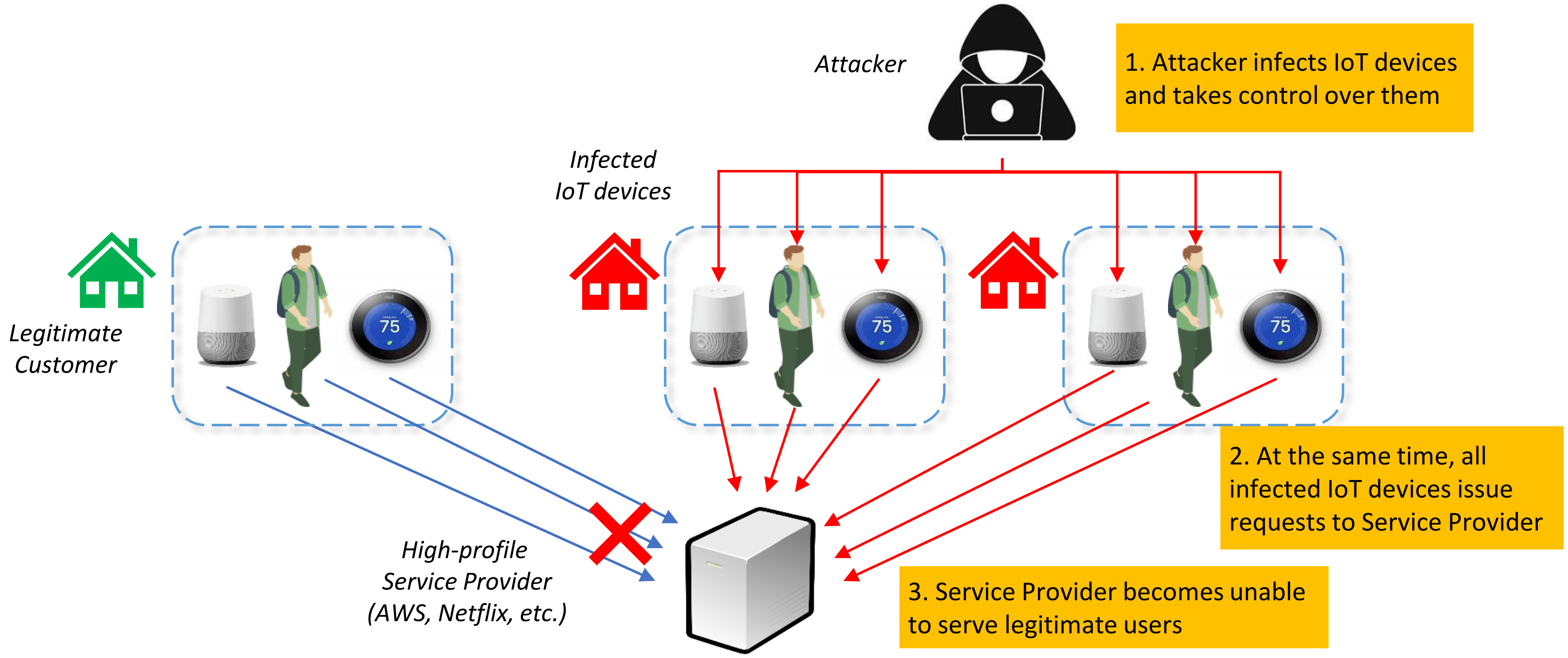
2



1. From https://www.elprocus.com/microcontrollers-types-and-applications/ . Author unknown. 2. From https://pt.farnell.com/raspberry-pi/rpi4-modbp-4gb/raspberry-pi-4-model-b-4gb/dp/3051887 . Author unknown.

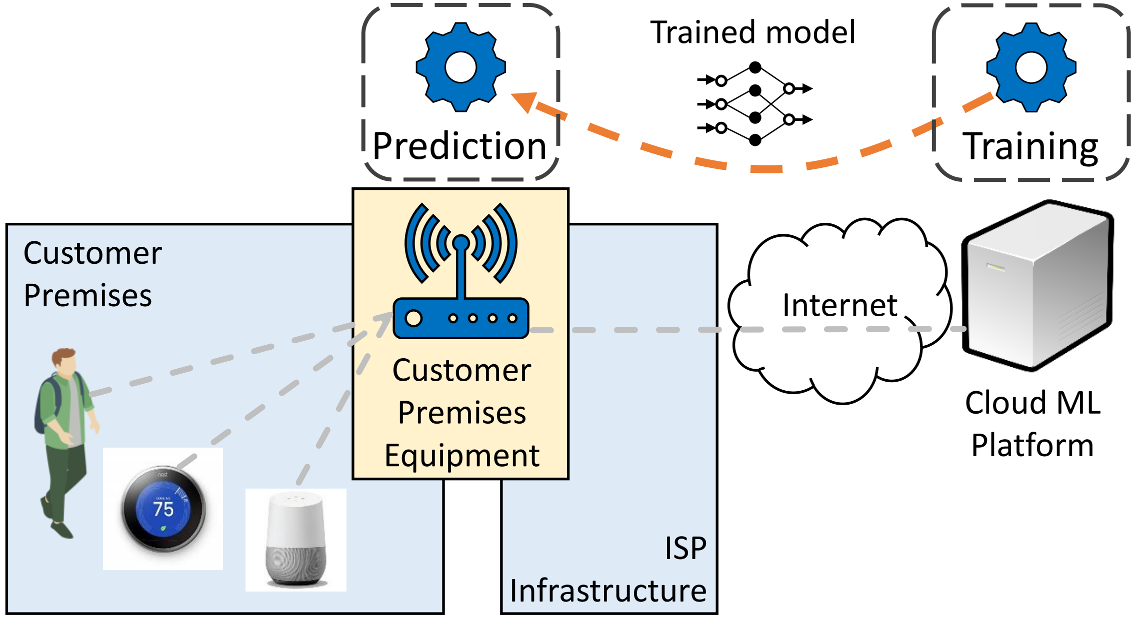
Motivational Use-case: Intrusion Detection

- Distributed Denial of Service (DDoS) attacks aim at disrupting the servers of high- profile online services (e.g., Amazon, Google or Netflix).
- To do so, a very large number of infected devices (typically vulnerable IoT devices) issues dummy requests to those servers.



Motivational Use-case: Intrusion Detection

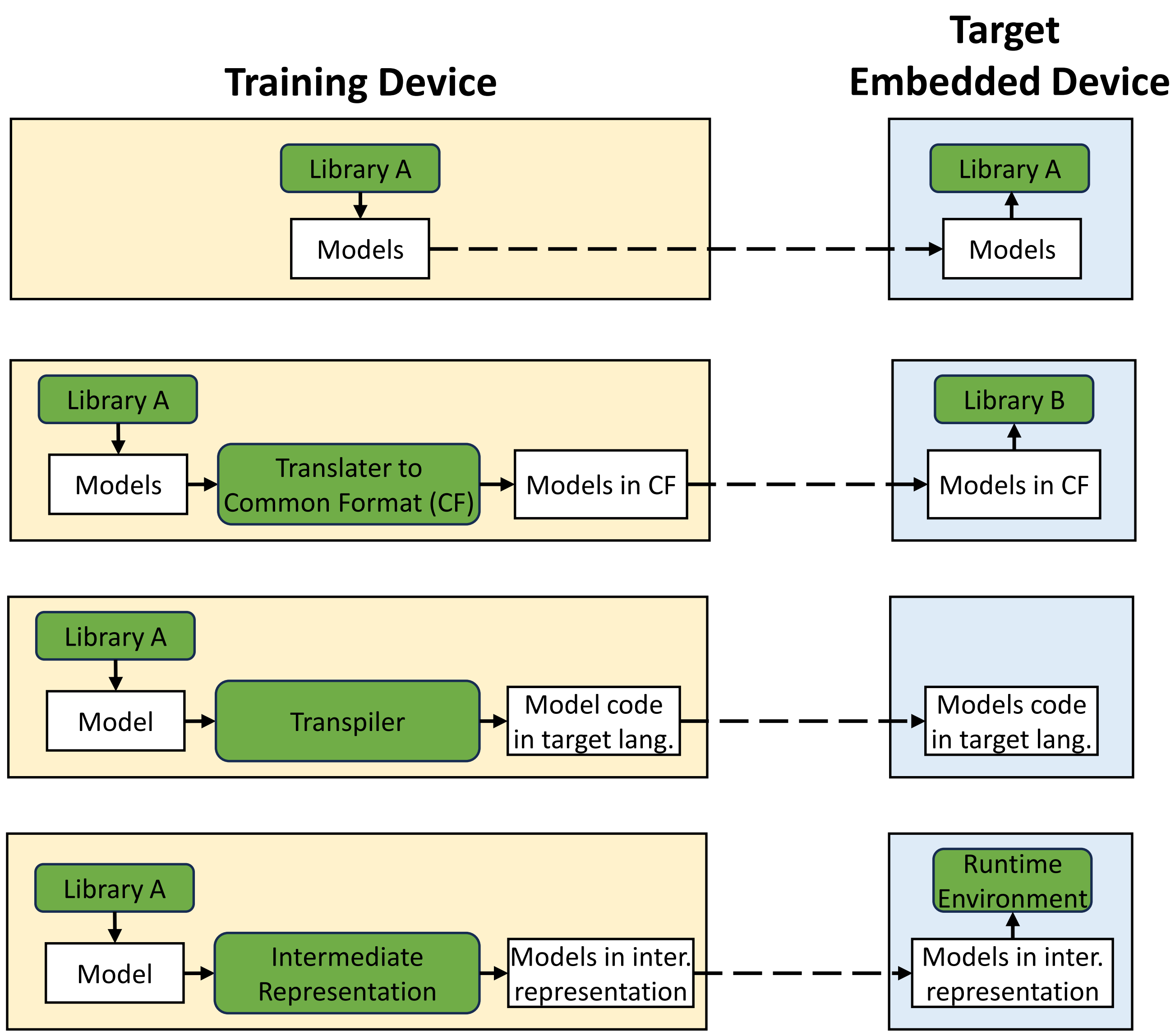
- Internet Service Providers (ISP) wish to mitigate cyberattacks through Intrusion Detection Systems (IDS).
- For an ISP, the IDS should be the closest possible to the targets: the Customer Premises Equipment (CPE).
- A common strategy is to **carry out training at the cloud** (due to the higher processing capabilities available), whereas the embedded device only performs prediction.



We are particularly interested in the scenario in which the models are trained using library A (e.g., at the cloud) and inference occurs in a different tool at the target device

Possible Approaches

- **ML libraries**
- **Interoperability standards**
- **Transpilers**
- **Runtime Environments**



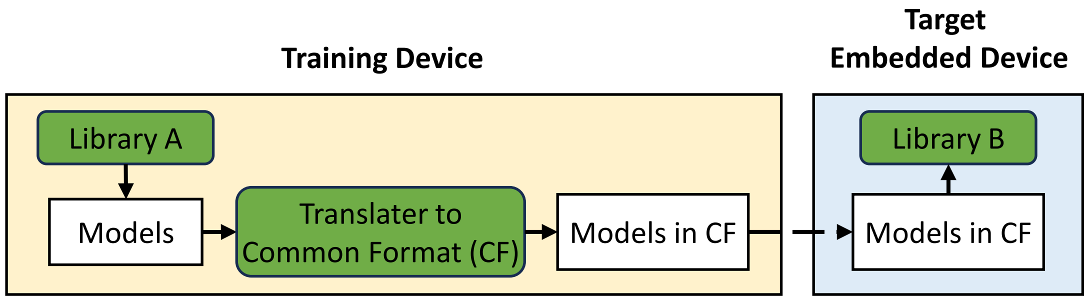
ML Libraries

Name	Written	Provided	Notes
TensorFlow	C++	Composed of datasets and pre-trained models developed and released by the Tensorflow Community	Colaboratory (Colab) for instance, is a free Jupyter notebook environment and runs in the cloud
Armadillo	C++	Linear algebra and scientific computing	It can use Open multi-processing (OpenMP), a free easy-to-use library for parallel computing.
Mlpack	C++	Fast and extensible implementations of ML models	Combination of Armadillo, ensmallen (a library for numerical optimization) and cereal (a serialization library).
Shogun	C++	Implements all the standard ML algorithms	Provides interfaces for C++, Python, Octave, R, Java, Lua, C#, Ruby
SHARK	C++	Neural networks, kernel-based learning algorithms, linear and nonlinear optimization methods	Shark works on Windows, MacOS X, and Linux
CAFFE	C++	Neural networks (e.g., CNN, RCNN, LSTM)	Mostly focused on deep learning using neural networks

- However, we observed that, despite being described in C/C++, most of these libraries do not seem tailored for deployment in resource-constrained devices

Interoperability Formats & Tools

Standards to provide a common description of ML models, therefore enabling porting between libraries.



Selected Tools

- **Open Neural Network Exchange (ONNX)**
  - ONNX is an open format built to represent machine learning models.
  - It defines a common set of operators - the building blocks of machine learning and deep learning models.
  - ONNX is compatible with at least 29 frameworks and converters and 30 inference runtimes.
- **Predictive Model Markup Language (PMML)**
  - Format based on Extensible Markup Language (XML) that can be used to described machine learning algorithms.
  - It enables ML model porting between existing supporting libraries and languages
    - C++: *cPMML*
    - Python/Scikit-Learn library: *sklearn2pmml*

A Glance into ONNX Format

In a glance:

- A code expression can be represented as a graph.
- Building an ONNX graph means implementing a function with the ONNX operators.

In more detail:

- ONNX is an open specification with the following components:
  - a definition of an extensible computation graph model;
  - definitions of standard data types;
  - and definitions of built-in operators.

  ONNX Intermediate Representation (or IR)

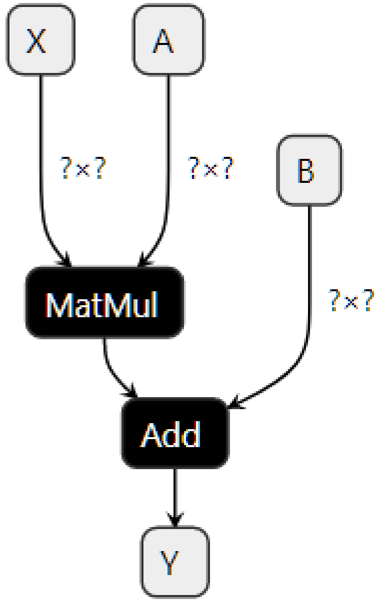
- In ONNX IR, each computation dataflow graph is structured as a list of nodes that form an acyclic graph. Each node is a call to an operator.

- Every framework supporting ONNX implements these operators on the applicable data types.

Linear regression as a code expression

```
def onnx_linear_regressor(X):
    "ONNX code for a linear regression"
    return onnx.Add(onnx.MatMul(X, coefficients), bias)
```

Linear regression as a graph



Example taken from: https://onnx.ai/onnx/intro/concepts.html

A Glance into PMML

List of models available

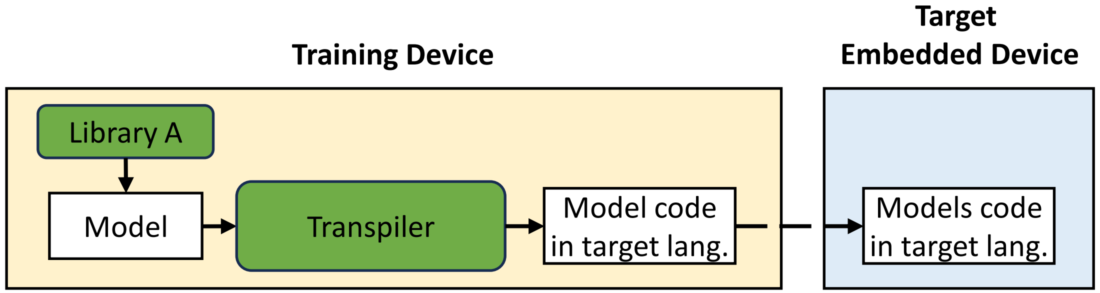
```
<xs:group name="MODEL-ELEMENT">
  <xs:choice>
    <xs:element ref="AnomalyDetectionModel"/>
    <xs:element ref="AssociationModel"/>
    <xs:element ref="BayesianNetworkModel"/>
    <xs:element ref="BaselineModel"/>
    <xs:element ref="ClusteringModel"/>
    <xs:element ref="GaussianProcessModel"/>
    <xs:element ref="GeneralRegressionModel"/>
    <xs:element ref="MiningModel"/>
    <xs:element ref="NaiveBayesModel"/>
    <xs:element ref="NearestNeighborModel"/>
    <xs:element ref="NeuralNetwork"/>
    <xs:element ref="RegressionModel"/>
    <xs:element ref="RuleSetModel"/>
    <xs:element ref="SequenceModel"/>
    <xs:element ref="Scorecard"/>
    <xs:element ref="SupportVectorMachineModel"/>
    <xs:element ref="TextModel"/>
    <xs:element ref="TimeSeriesModel"/>
    <xs:element ref="TreeModel"/>
  </xs:choice>
</xs:group>
```

Example of a Neural Network:

```
<xs:element name="NeuralInput">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="DerivedField"/>
    </xs:sequence>
    <xs:attribute name="id" type="NN-NEURON-ID" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="Neuron">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element maxOccurs="unbounded" ref="Con"/>
    </xs:sequence>
    <xs:attribute name="id" type="NN-NEURON-ID" use="required"/>
    <xs:attribute name="bias" type="REAL-NUMBER"/>
    <xs:attribute name="width" type="REAL-NUMBER"/>
    <xs:attribute name="altitude" type="REAL-NUMBER"/>
  </xs:complexType>
</xs:element>

<xs:element name="Con">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="from" type="NN-NEURON-IDREF" use="required"/>
    <xs:attribute name="weight" type="REAL-NUMBER" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="NeuralOutput">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Extension" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="DerivedField"/>
    </xs:sequence>
    <xs:attribute name="outputNeuron" type="NN-NEURON-IDREF" use="required"/>
  </xs:complexType>
</xs:element>
```

Transpilers

Transpilers translate a source code into
a language different than the original one.



Selected Tools

- **Sklearn-porter**: a FOSS Python library developed to transpile ML models from Scikit-Learn to other programming languages (C, Java, GO and JavaScript).
  https://github.com/nok/sklearn-porter
- **Model 2 Code Generator (m2cgen):** is a FOSS library developed in Python, that transpiles trained statistical models (trained, e.g., with Scikit-learn or lightning libraries) into a native code for at least 16 different programming languages (R, Visual Basic, Haskell, C#, etc.).
  https://github.com/BayesWitnesses/m2cgen

But… how do they work?

- Little or no documentation! Need to get into code

Two Examples from m2cgen

Decision Tree example, transpiled to C:

```
#include <string.h>
void score(double * input, double * output) {
    double var0[3];
    if (input[2] <= 2.449999988079071) {
        memcpy(var0, (double[]){1.0, 0.0, 0.0}, 3 * sizeof(double));
    } else {
        if (input[3] <= 1.75) {
            if (input[2] <= 4.950000047683716) {
                if (input[3] <= 1.6500000357627869) {
                    memcpy(var0, (double[]){0.0, 1.0, 0.0}, 3 * sizeof(double));
                } else {
                    memcpy(var0, (double[]){0.0, 0.0, 1.0}, 3 * sizeof(double));
                }
            } else {
                memcpy(var0, (double[]){0.0, 0.3333333333333333, 0.6666666666666666}, 3 * sizeof(double));
            }
        } else {
            memcpy(var0, (double[]){0.0, 0.021739130434782608, 0.9782608695652174}, 3 * sizeof(double));
        }
    }
    memcpy(output, var0, 3 * sizeof(double));
}
```

SVM example, transpiled to C:

```
#include <math.h>
#include <string.h>
void score(double * input, double * output) {
    double var0;
    var0 = exp(-0.06389634699048878 * (pow(5.1 - input[0], 2.0) + pow(2.5 - input[1], 2.0) + pow(3.0 - input[2], 2.0) + pow(1.1 - input[3], 2.0)));
    double var1;
    var1 = exp(-0.06389634699048878 * (pow(4.9 - input[0], 2.0) + pow(2.4 - input[1], 2.0) + pow(3.3 - input[2], 2.0) + pow(1.0 - input[3], 2.0)));
    ...
    var27 = exp(-0.06389634699048878 * (pow(6.3 - input[0], 2.0) + pow(2.8 - input[1], 2.0) + pow(5.1 - input[2], 2.0) + pow(1.5 - input[3], 2.0)));
    memcpy(output, (double[]){0.11172510039290856 + var0 * -0.8898986041811555 + var1 * -0.8898986041811555 + var2 * -0.0 + var3 * -0.0 +
    ...
    var5 * 0.0 + var6 * 110.34516826676301 + var7 * 0.0 + var8 * 110.34516826676301 + var9 * 110.34516826676301 + var10 * 0.0}, 3 * sizeof(double));
}
```

Transpilers

- Limited set of conversions

Sklearn-porter:

Estimator	Programming language					
Classifier	Java *	JS	C	Go	PHP	Ruby
svm.SVC	✓, ✓^{I}	✓	✓		✓	✓
svm.NuSVC	✓, ✓^{I}	✓	✓		✓	✓
svm.LinearSVC	✓, ✓^{I}	✓	✓	✓	✓	✓
tree.DecisionTreeClassifier	✓, ✓^{E}, ✓^{I}	✓, ✓^{E}	✓, ✓^{E}	✓, ✓^{E}	✓, ✓^{E}	✓, ✓^{E}
ensemble.RandomForestClassifier	✓^{E}, ✓^{I}	✓^{E}	✓^{E}	✓^{E}	✓^{E}	✓^{E}
ensemble.ExtraTreesClassifier	✓^{E}, ✓^{I}	✓^{E}	✓^{E}		✓^{E}	✓^{E}
ensemble.AdaBoostClassifier	✓^{E}, ✓^{I}	✓^{E}, ✓^{I}	✓^{E}			
neighbors.KNeighborsClassifier	✓, ✓^{I}	✓, ✓^{I}				
naive_bayes.GaussianNB	✓, ✓^{I}	✓				
naive_bayes.BernoulliNB	✓, ✓^{I}	✓				
neural_network.MLPClassifier	✓, ✓^{I}	✓, ✓^{I}				
Regressor	Java *	JS	C	Go	PHP	Ruby
neural_network.MLPRegressor		✓				

https://github.com/nok/sklearn-porter

m2cgen:

Supported Languages

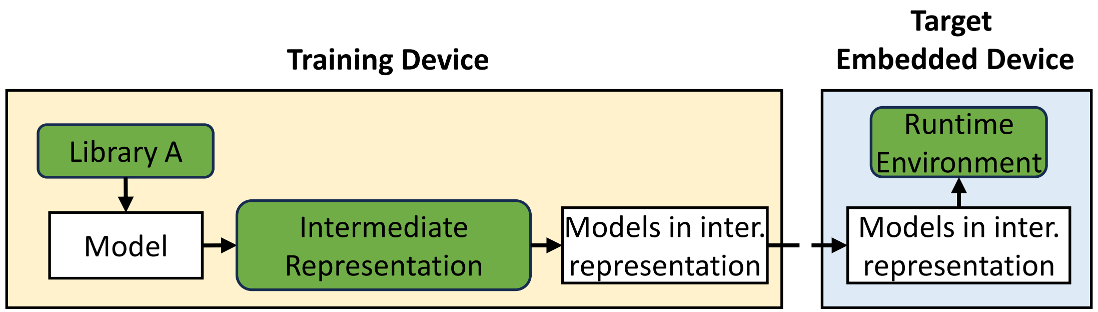
- C
- C#
- Dart
- F#
- Go
- Haskell
- Java
- JavaScript
- PHP
- PowerShell
- Python
- R
- Ruby
- Rust
- Visual Basic (VBA-compatible)
- Elixir

	Classification
Linear	• scikit-learn ∘ LogisticRegression ∘ LogisticRegressionCV ∘ PassiveAggressiveClassifier ∘ Perceptron ∘ RidgeClassifier ∘ RidgeClassifierCV ∘ SGDClassifier • lightning ∘ AdaGradClassifier ∘ CDClassifier ∘ FistaClassifier ∘ SAGAClassifier ∘ SAGClassifier ∘ SDCAClassifier ∘ SGDClassifier
SVM	• scikit-learn ∘ LinearSVC ∘ NuSVC ∘ OneClassSVM ∘ SVC • lightning ∘ KernelSVC ∘ LinearSVC
Tree	• DecisionTreeClassifier • ExtraTreeClassifier
Random Forest	• ExtraTreesClassifier • LGBMClassifier(rf booster only) • RandomForestClassifier • XGBRFClassifier
Boosting	• LGBMClassifier(gbdt/dart/goss booster only) • XGBClassifier(gbtree(including boosted forests)/gblinear booster only)

https://github.com/BayesWitnesses/m2cgen

Runtime Environments

Runtime Environments (RTE) execute (trained) ML models described in an intermediate description language.



Selected Tools

- **ONNX Runtime:** cross-platform machine-learning model accelerator, used to deploy ONNX format models into production.
- **Tensorflow Lite:** TF-variant tailored for resource-constrained systems that also uses a runtime.

Discussion & Exploration

Name	Pros	Cons
ML Libraries (+IF)	• Full ability of training and inference	• Libraries may be large • May have dependencies
Transpiler	• Stand-alone solution: no dependencies at target device	• Typically support only a limited set of models • Not much documentation on what's happening
Runtime Environment (+IF)	• Lightweight dependency: avoids deploying the entire library at the target device	• Training may not be available at device running RTE • Set of models at disposal may be limited

Name	Pros	Cons
Interoperability Formats (IF)	• Explainable methodologies to store models	• Success depends on adoption by libraries

We experimented with:

- **PMML (Interoperability format)**: Sklearn models → *sklearn2pmml* → Models in PMML → *cPMML* → Models in C
  - Outcome: unable to execute final code
- **m2cgen (Transpiler):** Sklearn model (one-class SVM) → *mc2gen* → Model in C
  - Outcome: results were similar, but we could not understand code
- **ONNX (Interoperability format)**: Sklearn models → *skl2onnx* → Models in ONNX format
  - To Tensorflow: ONNX format → *onnx-tf* → TF model – **Export failed**
- **ONNX Runtime (RTE):** installation via Python and use of models in ONNX format was straightforward

Comparative Analysis

Models

- Anomaly detection models: **Isolation Forest** | **Local Outlier Factor** (LOF)| **One-class SVM** | **SGD One-class SVM**

Training

- Datasets: we used the publicly-available datasets described in Table I
- Original library: models trained using *Scikit-Learn*
- Converted to the ONNX specification using *sklearn-onnx*

TABLE I: Dataset descriptions.

Dataset	Traffic type	# samples	# Features
IOT23 [15]	IoT devices	487	26
Botnet [16]	Data theft	196	26

Platforms

- Table II presents the characteristics of the selected computing platforms.

TABLE II: Selected platforms.

	Desktop	Raspberry Pi
Number of cores	4	4
Frequency utilized	2.00 GHz	600.00 MHz
RAM memory	9.64 GB	1.91 GB
Operating System	Ubuntu 20.04.6 LTS	Debian GNU/Linux 11
Python version	3.8.10	3.9.2
ONNX version	1.13.1	N/A
ONNXRuntime	1.14.1	1.14.1

Selected Tools

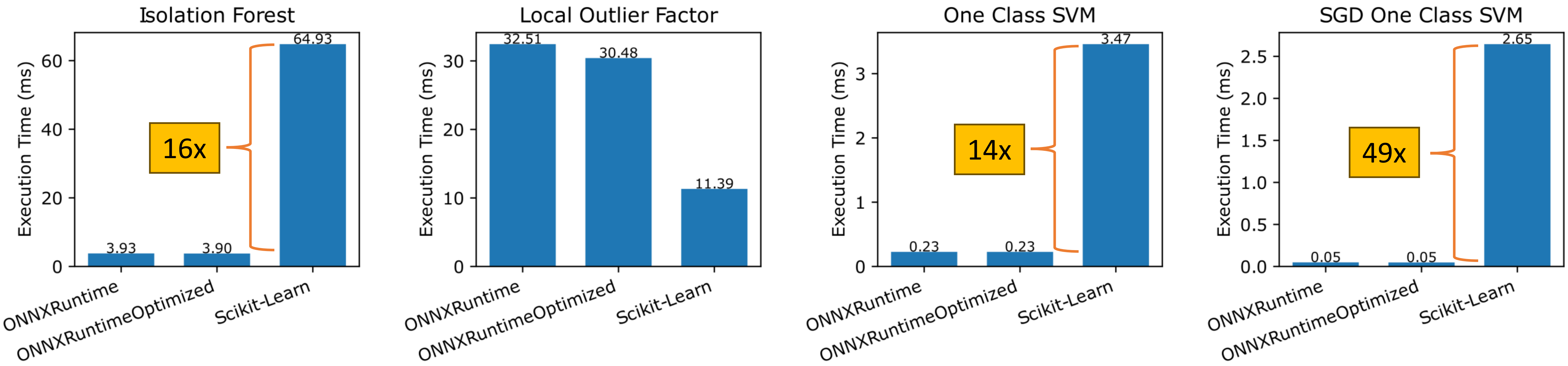
- **ONNX Runtime:** the ML tool selected as the most promising to evaluate
- **Scikit-Learn:** serves as benchmark (widely-used and used to train the models)
- **ONNX Runtime Optimized:** a variant that optimizes the ONNX graphs describing the models

Results on Desktop

We observe that *ONNX Runtime* accelerates all most models:

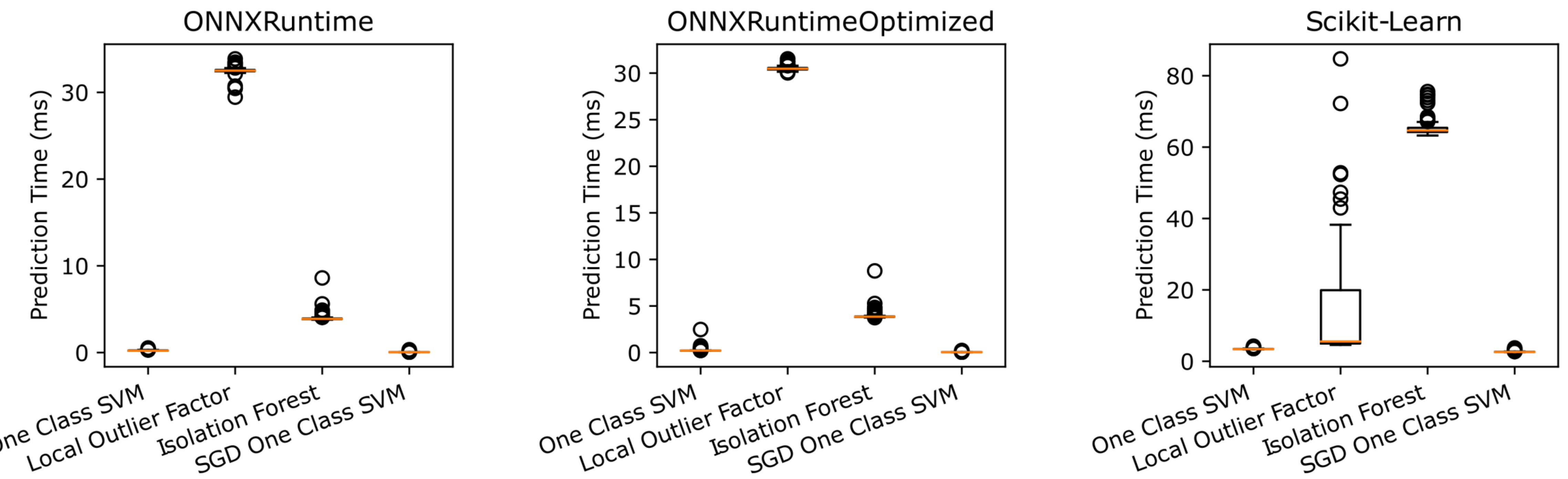
- ≈ 16x for Isolation Forest
- ≈ 14x for OC-SVM
- ≈ 49x for SGDSVM.

Notably, this is not observed for Local Outlier Factor (LOF), taking longer than *Scikit-learn*.



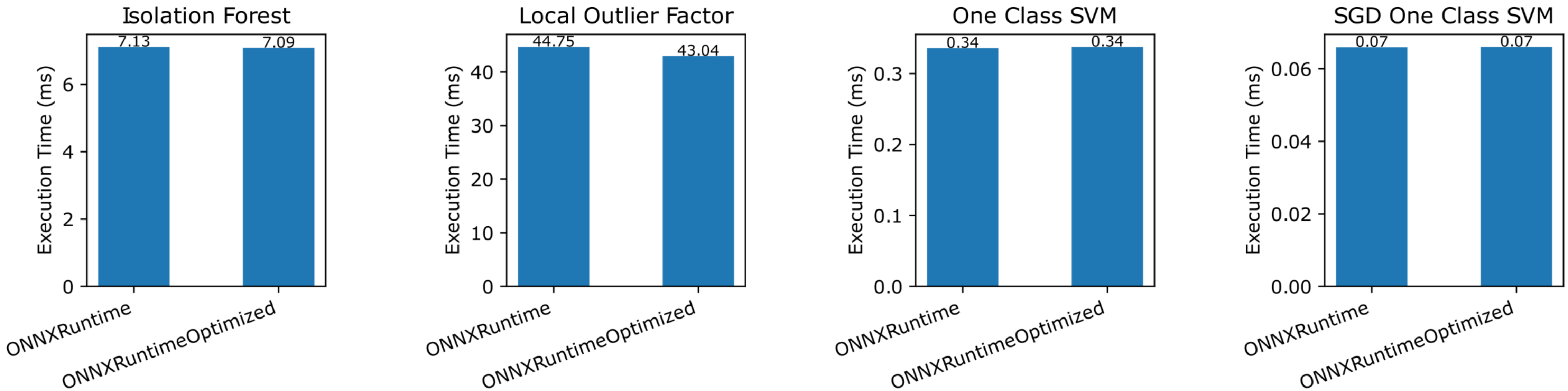
Results on Desktop

- Distribution of time execution per tool and model



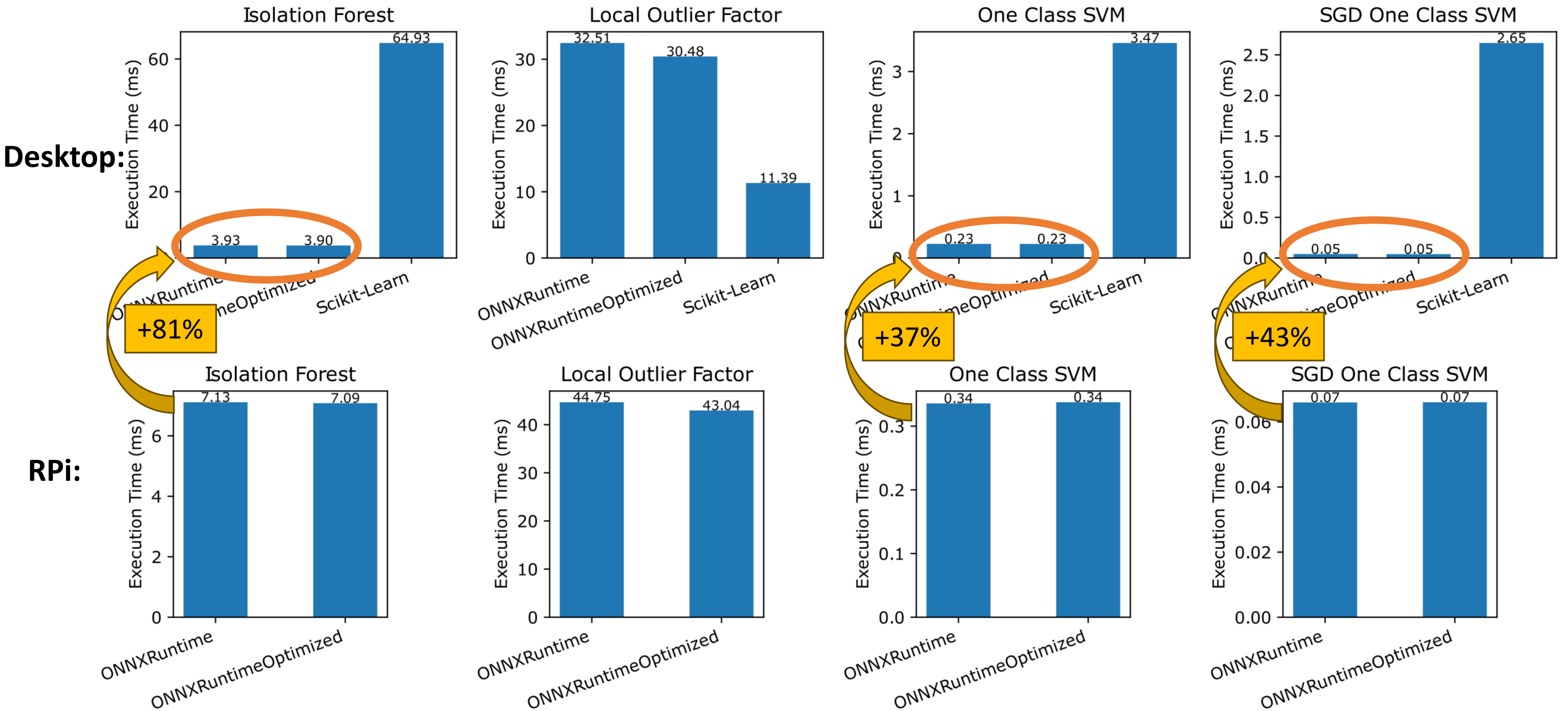
Results on Raspberry Pi

RPi:



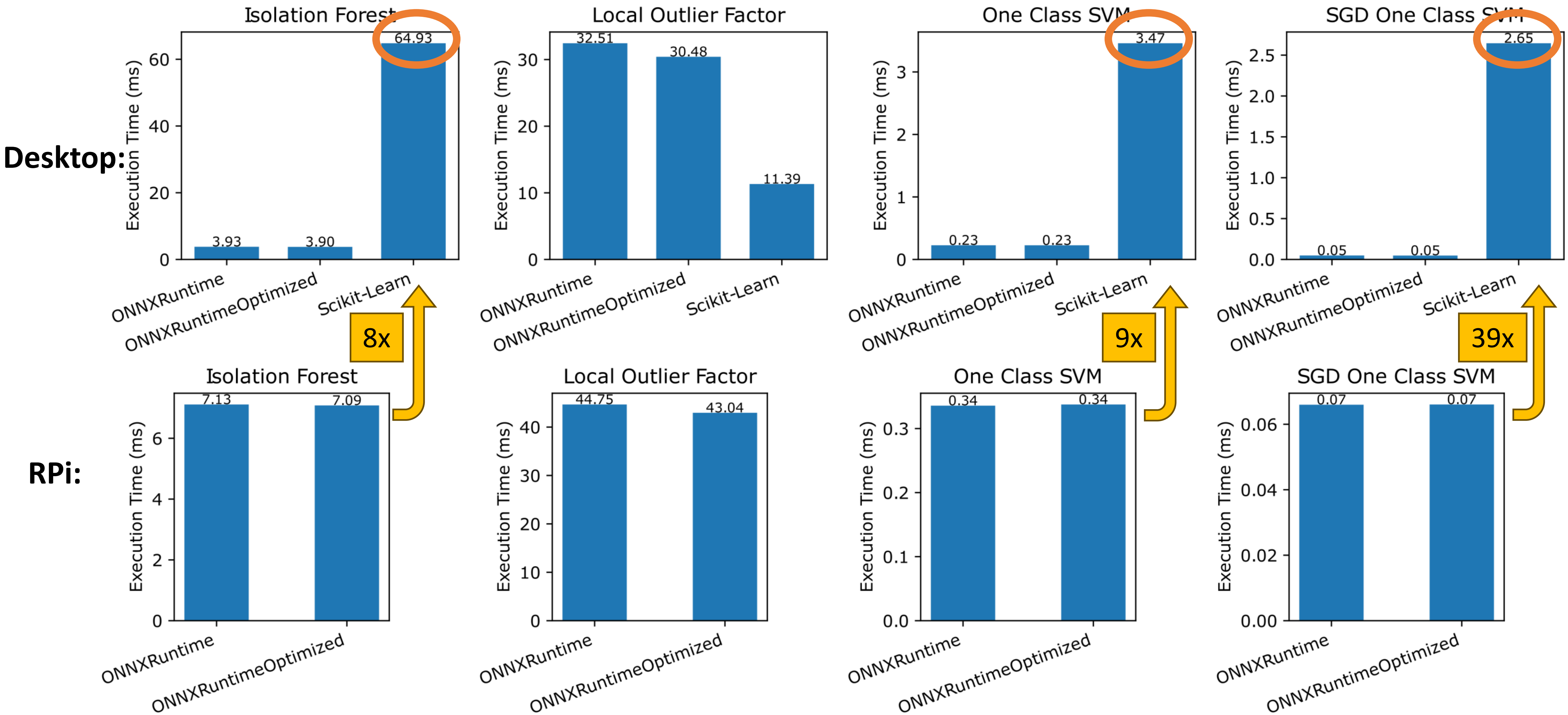
Results on Raspberry Pi

ONNX Runtime at Desktop and RPi



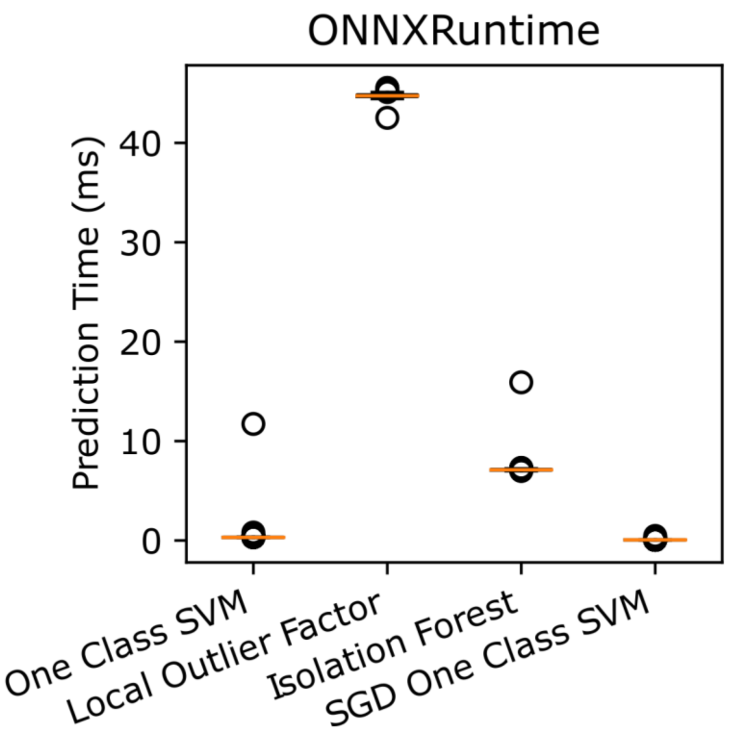
Results on Raspberry Pi

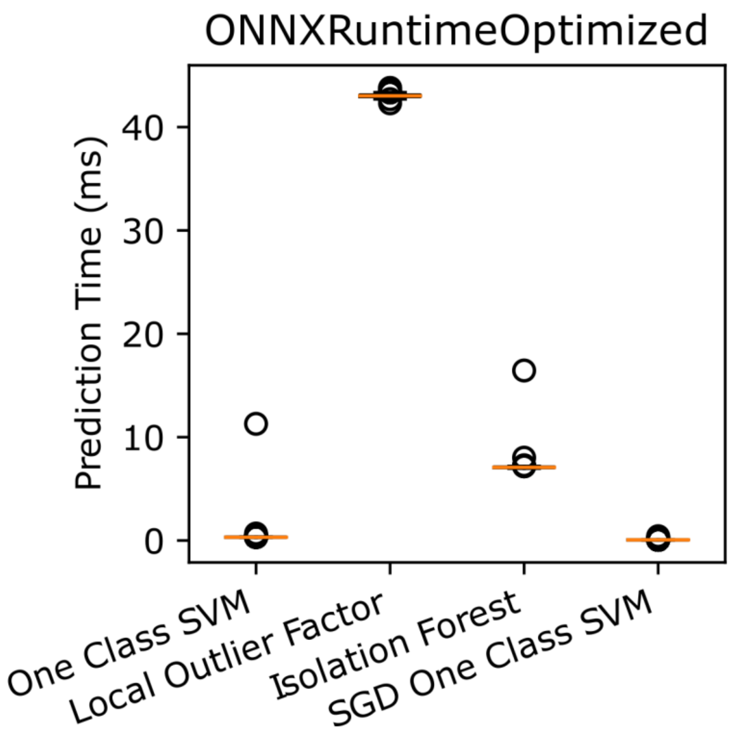
ONNX Runtime at RPi vs Scikit-Learn at Desktop



Results on Raspberry Pi

- Distribution of time execution per tool and model





Conclusions

- We reviewed Machine Learning (ML) tools according to their potential for embedded system.
- We selected a particular tool, **ONNX Runtime**, for comparing prediction time against the well-established Python-based ***Scikit-Learn***.
- The prediction time was measured in two platforms – a **standard desktop** and a target embedded system, a **Raspberry Pi v4** – for four pre-trained ML models and datasets.
- We observe that **ONNX Runtime considerably improves over the prediction time of Scikit-Learn**, and experiences a negligible performance degradation when ported to the RPi.
- Future work will continue this analysis over more ML tools and platforms, and adjusting model target accuracy.