Machine Learning Models Deployment using ML2CPP framework

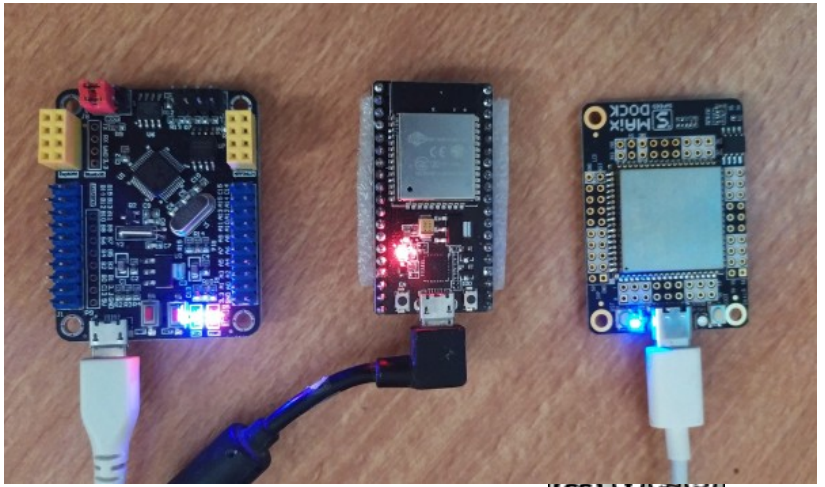
Technical Presentation

Antoine CARME (2022)
Antoine.CARME@outlook.com

https://github.com/antoinecarme/ml2cpp

What is ML2CPP ?

- ML2CPP is a “development tool” for generating deployment C++ code from Machine Learning models.
  - https://github.com/antoinecarme/ml2cpp
- Using ML2CPP, it is possible to predict values from an already-fitted classifier or a regressor on cheap widely-available hardware simply by executing some compiled C++ code.
- C++ is the most hardware-friendly programming language. Compilers are available on most platforms (GCC).
- ML2CPP allows for example to deploy an XGBoost, scikit-learn or a caret model
- It generates binary executables for
  - Small Edge controllers like STM32 or esp32,
  - Android, iOS, Linux, Windows devices,
  - But also for standard cloud virtual machines at the highest possible speed.



Supported Machine Learning Models

- ML2CPP shares the same low-level framework as Sklearn2SQL , a SQL generator for machine learning models used for in-database deployment.
  - It is written in python.
  - https://github.com/antoinecarme/sklearn2sql-demo
  - Some of these slides are copies of the same presentation for Sklearn2SQL (module SQL => C++)
- It is designed to support all classification and regression methods in scikit-learn
  - SVMs, linear models, naive-bayes. decision trees, MLP, etc
  - Transformers (PCA, imputers, scalers, ...)
  - Feature selection
  - Outlier detection
  - Derived models (random forests, meta-estimators, pipelines, feature unions, ensembles, etc).

Some supported Hardware

Chip	Arch	Bits	Cores	Speed	Memory	FPU
K210	riscv (IMAFDC)	64	2	400	8MB	2
ESP32D0WDQ6	xtensa	32	2	240	4MB	1
STM32F103C8T6	ARM Cortex-M3	32	1	72	64K	0

Generated C++ Code

- The C++ code contains everything needed to compute the predicted values of the model,
- No external library is needed,
- Can be compiled for any target hardware platform using any standard C++ compiler on the market (GCC).

- The automatically generated code is plain STL C++-17,
- Designed to maintain a strong semantic mapping with the model and allows auditing , debugging, benchmarking and reporting.

- The generated code does/will not rely on any kind of external library (not use BLAS or tensorflow or ...) and will not need compiler specific flags.
- It is generated such that any C++ code can integrate it without specific change (the model code is a namespace).

Sample Codes 1/2

```cpp
// ********************************************************

// This C++ code was automatically generated by ml2cpp (development version).
// Copyright 2020

// https://github.com/antoinecarme/ml2cpp
// Model : DecisionTreeClassifier
// Dataset : iris

// This CPP code can be compiled using any C++-17 compiler.
// g++ -Wall -Wno-unused-function -std=c++17 -g -o ml2cpp-demo_DecisionTreeClassifier_iris.exe ml2cpp-demo_DecisionTreeClassifier_iris.cpp

// Model deployment code

// ********************************************************

#include "../../Generic.i"

namespace  {

        std::vector<std::any> get_classes(){
                std::vector<std::any> lClasses = { 0, 1, 2 };

                return lClasses;
        }

        typedef std::vector<double> tNodeData;
        std::map<int, tNodeData> Decision_Tree_Node_data = {
                        { 1 ,  {1.0, 0.0, 0.0 }} ,
                        { 4 ,  {0.0, 1.0, 0.0 }} ,
                        { 6 ,  {0.0, 0.0, 1.0 }} ,
                        { 7 ,  {0.0, 1.0, 0.0 }} ,
                        { 11 ,  {0.0, 0.0, 1.0 }} ,
                        { 12 ,  {0.0, 1.0, 0.0 }} ,
                        { 13 ,  {0.0, 0.0, 1.0 }} ,
                        { 14 ,  {0.0, 0.0, 1.0 }}
        };


        int get_decision_tree_node_index(std::any Feature_0, std::any Feature_1, std::any Feature_2, std::any Feature_3) {
                int lNodeIndex = (Feature_3 <= 0.800000011920929) ? ( 1 ) : ( (Feature_2 <= 4.8500001430511475) ? ( (Feature_3 <= 1.6500000357627869) ? ( 4

                return lNodeIndex;
        }


        std::vector<std::string> get_input_names(){
                std::vector<std::string> lFeatures = { "Feature_0", "Feature_1", "Feature_2", "Feature_3" };

                return lFeatures;
        }

        std::vector<std::string> get_output_names(){
                std::vector<std::string> lOutputs = {
                        "Score_0", "Score_1", "Score_2",
                        "Proba_0", "Proba_1", "Proba_2",
                        "LogProba_0", "LogProba_1", "LogProba_2",
                        "Decision", "DecisionProba" };

                return lOutputs;
        }

        tTable compute_classification_scores(std::any Feature_0, std::any Feature_1, std::any Feature_2, std::any Feature_3) {
                auto lClasses = get_classes();

                int lNodeIndex = get_decision_tree_node_index(Feature_0, Feature_1, Feature_2, Feature_3);

                std::vector<double> lNodeValue = Decision_Tree_Node_data[ lNodeIndex ];


                tTable lTable;

                lTable["Score"] = {
                        std::any(),
                        std::any(),
                        std::any()
                } ;
                lTable["Proba"] = {
                        lNodeValue [ 0 ],
                        lNodeValue [ 1 ],
                        lNodeValue [ 2 ]
                } ;
                int lBestClass = get_arg_max( lTable["Proba"] );
                auto lDecision = lClasses[lBestClass];
                lTable["Decision"] = { lDecision } ;
                lTable["DecisionProba"] = { lTable["Proba"][lBestClass] };

                recompute_log_probas( lTable );

                return lTable;
        }

        tTable compute_model_outputs_from_table( tTable const & iTable) {
                tTable lTable = compute_classification_scores(iTable.at("Feature_0")[0], iTable.at("Feature_1")[0], iTable.at("Feature_2")[0], iTable.at("Fea

                return lTable;
        }

} // eof namespace


int main() {
        score_csv_file("outputs/ml2cpp-demo/datasets/iris.csv");
        return 0;
}
```

Sample Codes 2/2

https://github.com/antoinecarme/ml2cpp/tree/master/bench/ml2cpp-demo/XGBClassifier/FourClass_100

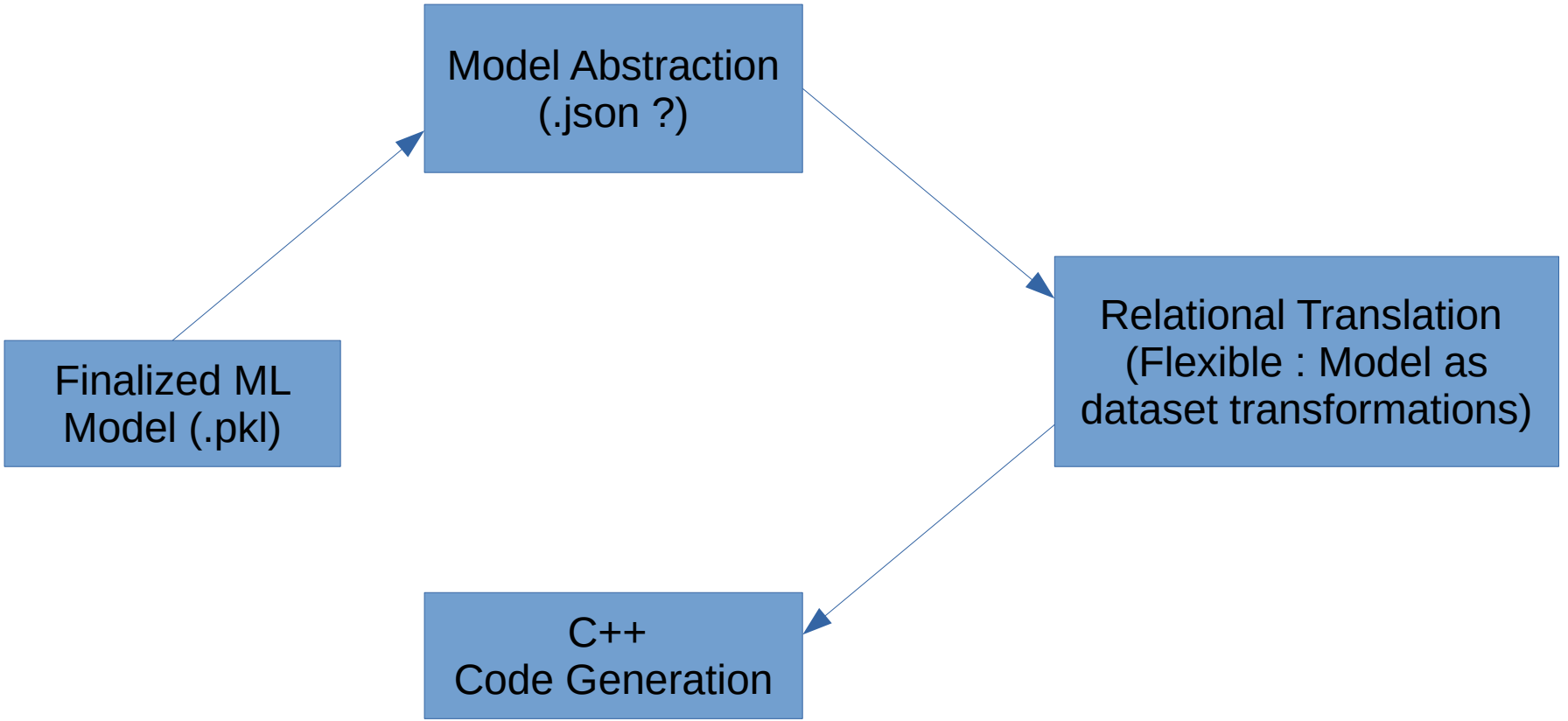
```
// https://github.com/antoinecarme/ml2cpp
// Model : XGBClassifier
// Dataset : FourClass_100

// This CPP code can be compiled using any C++-17 compiler.
// g++ -Wall -Wno-unused-function -std=c++17 -g -o ml2cpp-demo_XGBClassifier_FourClass_100.exe ml2cpp-demo_XGBClassifier_FourClass_100.cpp
```

```
17  #include "../../Generic.i"
18
19  namespace  {
20
21          std::vector<std::any> get_classes(){
22                  std::vector<std::any> lClasses = { 0, 1, 2, 3 };
23
24                  return lClasses;
25          }
26
27          namespace XGB_Tree_0_0 {
28
29                  std::vector<std::any> get_classes(){
30                          std::vector<std::any> lClasses = { 0, 1, 2, 3 };
31
32                          return lClasses;
33                  }
34
35                  typedef std::vector<double> tNodeData;
36                  std::map<int, tNodeData> Decision_Tree_Node_data = {
37                                  { 1 ,  {-0.156164378 }} ,
38                                  { 2 ,  {0.144303814 }}
39                  };
40
41
42                  int get_decision_tree_node_index(std::any Feature_0, std::any Feature_1, s
43                          int lNodeIndex = ( Feature_78 < -0.590290248 ) ? ( 1 ) : ( 2 );
44
45                          return lNodeIndex;
46                  }
```

```
4865                            XGB_Tree_2_13::compute_classification_scores(Feature_0, Feature_
4866                            XGB_Tree_3_13::compute_classification_scores(Feature_0, Feature_
4867                            XGB_Tree_0_14::compute_classification_scores(Feature_0, Feature_
4868                            XGB_Tree_1_14::compute_classification_scores(Feature_0, Feature_
4869                            XGB_Tree_2_14::compute_classification_scores(Feature_0, Feature_
4870                            XGB_Tree_3_14::compute_classification_scores(Feature_0, Feature_
4871                            XGB_Tree_0_15::compute_classification_scores(Feature_0, Feature_
4872                            XGB_Tree_1_15::compute_classification_scores(Feature_0, Feature_
4873                            XGB_Tree_2_15::compute_classification_scores(Feature_0, Feature_
4874                            XGB_Tree_3_15::compute_classification_scores(Feature_0, Feature_
4875                    };
4876
4877                    tTable lAggregatedTable = aggregate_xgb_scores(lTreeScores, {"Score"});
4878
4879                    tTable lSoftMaxTable = soft_max(lAggregatedTable);
4880
4881
4882                    tTable lTable = lSoftMaxTable;
4883
4884                    int lBestClass = get_arg_max( lTable["Proba"] );
4885                    auto lDecision = lClasses[lBestClass];
4886                    lTable["Decision"] = { lDecision } ;
4887                    lTable["DecisionProba"] = { lTable["Proba"][lBestClass] };
4888
4889                    recompute_log_probas( lTable );
4890
4891                    return lTable;
4892            }
```

Framework Description



System Input

- The system generates C++ code from already trained models.
- The gory details of the training process are not significant.
- Almost all public interfaces and web services take a serialized model as input (pickle is your friend here).

Model Abstraction

- In this step, the system performs a complete formal abstraction of the underlying algorithm of the model.
  - For a linear Model :
    - {“Type” : “linear”, “coefficients” : [8.88, 8888e-8], “intercept” : 888888}
- More complex formal abstractions are available for each model/algorithm type (DT, XGB, SVM, RF, Pipeline, ...).
- The abstraction must be complete, For a random forest (RF) model, the abstraction contains the individual abstractions of all the forest decision trees.
- Adding a new mathematical model requires designing/authoring these abstractions.
  - The math behind scikit-learn algorithms (and machine learning in general) is evolving slowly (need many years to “invent” a new model). Almost all of these models date back to 19xx.
    - Scikit-learn has a lot of requirements on what model can be added (popularity, publications, citations, impact,), and that’s really good. https://scikit-learn.org/stable/faq.html
  - The current version of the sklearn2sql framework has designed (> 40) abstractions of almost all scikit-learn models known before 2020.
    - https://scikit-learn.org/stable/modules/classes.html
- These abstractions are deduced from a reloaded/living model. We only use model metadata. The original dataset is not needed even if the model can help generating/simulating one.
- The formal abstraction is expressed in JSON format.
  - Almost all python objects have a __dict__ member. Python object introspection is easy.
  - Scikit Learn Models are aggregates of numpy python objects.
  - XGB models already have a to_json method. Easy.
  - JSON code is debuggable.

Relational Model Representation

- All machine learning data processing can be performed using sequences of simple dataset/columns transformations (dataset→dataset mapping).
- This translation is performed only using the “Complete” Model Abstraction (JSON format is enough).
- These dataset transformations can be translated into relations (as in a relational algebra / SQL).
- For a given model, The number of these transformations depends on the complexity of the model.
  - Think of these as the layers of a sequential neural network (Scikit MLP), but at a lower level.
- Individual transformations are developed using specialized python classes, with the most tasks done at the abstract level.
  - The transformation does not take into account the type of model it is part of.
  - Groups of transformations used to compute the predicted values (arg-max of probabilities) are shared across all model representations (the same C++ code between DTs and pipelines of SVMs).
- A scikit-learn machine learning model can be mapped this way.
  - A decision tree score computation can be performed by hand using ~4 excel sheets, and these sheets can be mapped to C++ classes.
  - A random forest with 500 trees, will be translated as ~4*500 separate relations. An additional relation is used to compute the RF class probabilities (means of 500 individual class probabilities).
    - Seems too mechanical. There are some technical C++ limitations but the usage of namespaces solves these code complexity issues.

C++ Code Generation 1/4

- C++ Code generation is about translation the model representation into a valid C++ code for a target compiler on the target platform.
- We use a symbolic hand-crafted generator as a framework for handling relational model representations internal transformations.
- Relations are generated in C++ as maps. It is common for holding all types of data (classifier inputs/outputs, regression inputs/outputs, transformer inputs/outputs, etc). A specific class is used
  - typedef std::vector<std::any> tAnyVector;
  - typedef std::map<std::string, tAnyVector> tTable;
- Generic methods are used to get the outputs of a model
  - tTable compute_model_outputs_from_table(tTable const & iTable)
- For controlling scopes, namespaces are used. A random forest model with 500 trees is generated as namespaces with 500 sub-namespaces.
  - For code readability, each internal namespace reflects the tree that is generated.
  - The use of namespaces makes life easier for the compiler/parser.

Extensions

- ML2CPP can be used to generate C++ code that is compiled to as part of a java or python project.
- We tested generating python module out of a Machine learning model using Boost. Python with a few extra lines of code.
  - https://www.boost.org/doc/libs/1_63_0/libs/python/doc/html/index.html
  - https://github.com/antoinecarme/ml2cpp/issues/35
- The model was trained using scikit-learn/numpy and has been deployed on a tiny micropython environment (where no scikit-learn numpy is available)
  - https://micropython.org/

Quality Assessment 1/2

- A test framework has been put in place.
- For each scikit-learn model class, we perform the following tests:
  - Train different models on datasets of various sizes (nrows, ncols). These datasets are stored as CSV files.
  - For each trained model :
    - save it,
    - Predict all the possible values for all the training dataset in python scikit-learn (sklearn_output_python_df)
    - generate the C++ code,
    - Compile and Execute the C++ code on the test platform (sparc debian T3-1) => CSV => pandas dataframes (gen_cpp_output_df).
    - Compare the relevant output columns between sklearn_output_python_df the output dataframe.
    - Add some reporting for success/failures each model.

谢谢 !!!!