NNPred C++ API Document

Weishuo Liu[a]

[a]*School of Energy and Power Engineering, Beihang University, 37 Xueyuan Road, Haidian District, Beijing, 100191, China*

A+B Model and Code Example

A simple **A + B = C** model is presented to show the deployment process. This model computes the sum of two 2D arrays (i.e., **A** and **B**) and holds the result in the output node, **C**. The I/O nodes' dimension sizes are $n \times 2$, with n being the number of data instances, and their data type is specified as the single-precision floating point. Figure. 1 shows a minimal application case of the model deployment and interaction with external arrays (i.e., `Array_A`, `Array_B` and `Array_C`). In this case, `Array_B` has different data type from the node's definition (`int` vs `float`), and `Array_C` differs to node, `output_c`, in both data type (`double` vs. `float`) and memory layout (3×2 vs. 3×2).

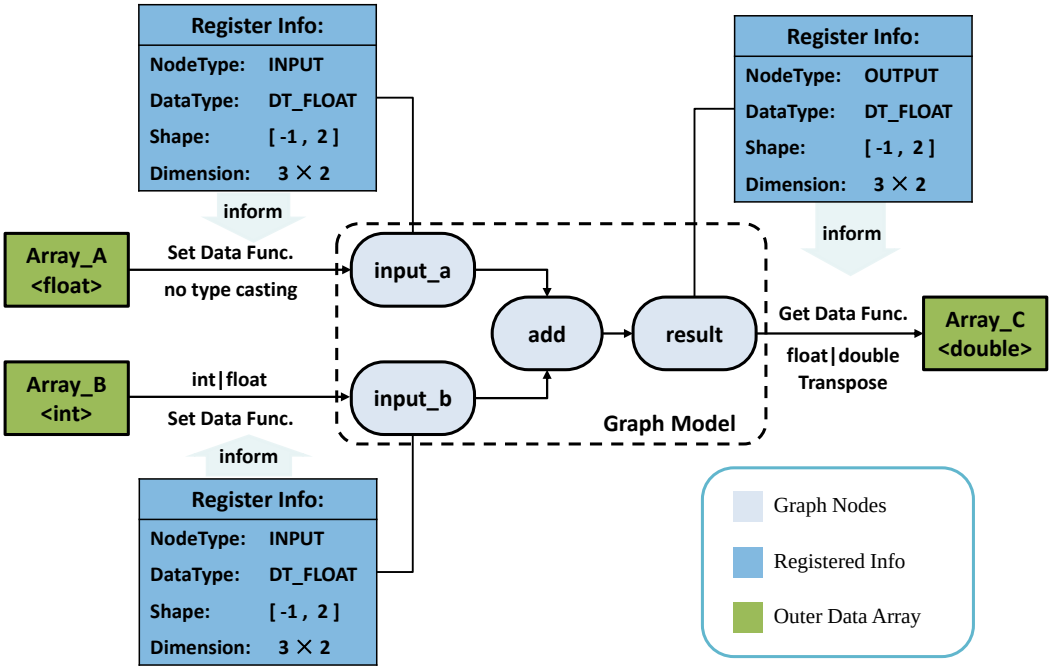

Figure 1: Interaction relation within NNPred elements and with outer memory spaces during a prediction process.

The code example to demonstrate the entire deploying process can be found in Listing 1. The whole process can be divided into two major parts: initialization and prediction. The initialization consists of loading the model, registering the input/output nodes, and setting the number of data instances, whereas the prediction part includes setting the input data, running predictions, and extracting the result. The location and function prototype of each step are listed in Table. 1.

Email address: `liuweishuo@buaa.edu.cn` (Weishuo Liu)

Table 1: ID, function prototypes and locations of necessary steps in C++

Step ID	Usage	Function Prototype *	Location
C01	Load Model	`Predictor(std::string PBfile)`	06-07
C02	Register Nodes	`regist_node(std::string node_name, Predictor::NodeType tp)`	09-16
C03	Set Data Counts	`set_data_count(int n_data)`	18-19
C04	Set Input Data	`set_node_data(std::string node_name, std::vector<T>& data)`	28-30
C05	Run Model	`run()`	32-33
C06	Get Output Data	`get_node_data(std::string node_name, std::vector<T>& data)`	35-36

* The function prototypes are all under public class: `class Predictor`. Therefore, the full reference of functions, for example for C01, should be: `Predictor::Predictor(std::string PBfile)`

```cpp
// Header files
#include "predictor.h"    // Predictor header
#include <vector>         // C++ standard header

int main(int argc, char const *argv[]) {
    // Load Model:
    Predictor pd("simple_graph_tf2.pb"); // Model's path or filename

    // Register node:
    // Inputs:
    // Predictor::INPUT_NODE is the node type enumerate
    pd.regist_node("input_a", Predictor::INPUT_NODE);
    pd.regist_node("input_b", Predictor::INPUT_NODE);
    // Outputs:
    // Predictor::OUTPUT_NODE is the node type enumerate
    pd.regist_node("result", Predictor::OUTPUT_NODE);

    // Set the number of data instances (n=3)
    pd.set_data_count(3);

    // Create external source of input/output data array:
    // Inputs:
    std::vector<float> vec_input1_float = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6};
    std::vector<int> vec_input2_float = {6, 5, 4, 3, 2, 1};
    // Outputs:
    std::vector<double> vec_out_float(6);

    // Set data for input nodes
    pd.set_node_data("input_a", vec_input1);
    pd.set_node_data("input_b", vec_input2);

    // Run model
    pd.run();

    // Get output into the target container
    pd.get_node_data("result", vec_out, Predictor::ColumnMajor);

    // Check results, expected calculation results:
    // vec_out: [7.1, 7.3, 7.5, 7.2, 7.4, 7.6]
    //          [C11, C21, C31, C12, C22, C32]
    return 0;
}
```

Listing 1: Minimal example to run A + B model in C++

In the initialization process, the node information (name, shape, and data type) is cached by the register-node step after the model is loaded. Once the number of data instances is set, the unknown dimension is determined and the internal buffers are created. In the prediction process, the set-data function will fill the input buffers, and the get-data function extracts the results after the prediction is executed. Meanwhile, the I/O functions could automatically cast the data type to get correct results, and the array can be mapped to a different memory layout with transpose options.

Entire API list for C++

- **C01: Loading Model:**
  - `Predictor(std::string pbfile)`
    - `pbfile` – the file name of the PB graph (i.e.,`simple_graph_tf2.pb`).

    Class constructor, to create the `Predictor` object from a `*.pb` format.
  - `Predictor(std::string folder, std::string tag)`
    - `folder` – the directory of the `SavedModel` format (this format itself is a folder).
    - `tags` – tags label within a `SavedModel` format, by default is `serve`[1].

    Class constructor, to create the `Predictor` object from a `SavedModel` format.
  - `Predictor(std::string pbfile, uint8_t para_intra, uint8_t para_inter)`
    - `pbfile` – the name of the PB graph (i.e.,`simple_graph_tf2.pb`).
    - `para_intra` – number of threads for internal parallelization (e.g., matrix multiplication and reduce sum).
    - `para_inter` – number of threads for operations independent with each other.

    Class constructor, to create the `Predictor` object from a `*.pb` format with parallelization configs. If both `para_intra` and `para_inter` are set to be **0**, the system will pick an appropriate number. If they are set to be **1**, the predictor will run model serially (an easy way to cooperate with MPI pattern in CFD codes).
  - `Predictor(std::string folder, std::string tag, uint8_t para_intra, uint8_t para_inter)`
    - `folder` – the directory of the `SavedModel` format (this format itself is a folder).
    - `tags` – tags label within a `SavedModel` format, by default is `serve`.
    - `para_intra` – number of threads for internal parallelization (e.g., matrix multiplication and reduce sum).
    - `para_inter` – number of threads for operations independent with each other.

    Class constructor, to create the `Predictor` object from a `SavedModel` format with parallelization configs. If both `para_intra` and `para_inter` are set to be **0**, the system will pick an appropriate number. If they are set to be **1**, the predictor will run model serially (an easy way to cooperate with MPI pattern in CFD codes).
- **C02: Register Nodes:**
  - `regist_node(std::string node_name, NodeType type)`
    - `node_name` – the name of the node to be registered as input or output.
    - `type` – the enumerate to specify node type, the available options are:
      - `Predictor::INPUT_NODE` – to register node as input.
      - `Predictor::OUTPUT_NODE` – to register node as output.

    To register input and output node for feeding and extracting data.
- **C03: Set Data Counts:**
  - `set_data_count(int n_data)`
    - `n_data` – the number of data instances.

    To set the number of data instances to substitute the unknown dimension of input/output tensors. For example, in the A + B example, the input/output shapes are all [-1, 2], the function could set the unknown -1 into concrete value so that the inner data containers can be created.
- **C04: Set Input Data:**
  - `set_node_data(std::string node_name, std::vector<T>& data)`
    - `node_name` – the name of the input node to be fed with external data.
    - `data` – the external data defined with C++ STL library (i.e., `std::vector`)

[1] Users could use the `saved_model_cli` to find out the tags in a `SavedModel` format

To feed the internal input data container registered under `node_name` with external data source hold by a C++ standard template library (STL) `std::vector`. If the data type of the internal container and external source are different, this function would automatically cast the datatype to resolve the difference.

- `set_node_data(std::string node_name, std::vector<T>& data, DataLayout layout)`
  - `node_name` – the name of the input node to be fed with external data.
  - `data` – the external data defined with C++ STL library (i.e., `std::vector`)
  - `layout` – the enumerate to specify whether the memory layout is transposed, the available options are:
    - `Predictor::RowMajor` – to hold the original memory sequence of the external data source.
    - `Predictor::ColumnMajor` – to perform matrix transpose while set data to internal container.

  To feed the internal input data container registered under `node_name` with external data source hold by a C++ standard template library (STL) `std::vector`. If the data type of the internal container and external source are different, this function would automatically cast the datatype to resolve the difference. The memory layout would be changed during the mapping process if `Predictor::ColumnMajor` is passed to the function.
- `set_node_data(std::string node_name, std::vector<T>& data, DataLayout layout, CopyMethod method)`
  - `node_name` – the name of the input node to be fed with external data.
  - `data` – the external data defined with C++ STL library (i.e., `std::vector`)
  - `layout` – the enumerate to specify whether the memory layout is transposed, the available options are:
    - `Predictor::RowMajor` – to hold the original memory sequence of the external data source.
    - `Predictor::ColumnMajor` – to perform matrix transpose while set data to internal container.
  - `method` – the enumerate to specify the copy method to the internal container:
    - `Predictor::Simple` – to copy/cast the arrays element-wise through C++ loop (possibly cache miss).
    - `Predictor::Eigen` – to copy/cast the arrays via `Eigen` library (SIMD is enabled).

  To feed the internal input data container registered under `node_name` with external data source hold by a C++ standard template library (STL) `std::vector`. If the data types of the internal container and external source are different, this function would automatically cast the datatype to resolve the difference. The memory layout would be changed during the mapping process if `Predictor::ColumnMajor` is passed to the function. The copy/cast procedure could be accelerated via `Eigen` library by specifying `Predictor::Eigen` options.
- `set_node_data(std::string node_name, T* p_data, int array_size)`
  - `node_name` – the name of the input node to be fed with external data.
  - `p_data` – the pointer to the first element of external data array.
  - `array_size` – the number of data elements of the external data array.

  To feed the internal input data container registered under `node_name` with external data source specified by the pointer to the array and the number of data elements. If the data types of the internal container and external source are different, this function would automatically cast the datatype to resolve the difference.
- `set_node_data(std::string node_name, T* p_data, int array_size, DataLayout layout)`
  - `node_name` – the name of the input node to be fed with external data.
  - `p_data` – the pointer to the first element of external data array.
  - `array_size` – the number of data elements of the external data array.
  - `layout` – the enumerate to specify whether the memory layout is transposed, the available options are:
    - `Predictor::RowMajor` – to hold the original memory sequence of the external data source.
    - `Predictor::ColumnMajor` – to perform matrix transpose while set data to internal container.

  To feed the internal input data container registered under `node_name` with external data source specified by the pointer to the array and the number of data elements. If the data types of the internal container and external source are different, this function would automatically cast the datatype to resolve the difference. The memory layout would be changed during the mapping process if `Predictor::ColumnMajor` is passed to the function.
- `set_node_data(std::string node_name, T* p_data, int array_size, DataLayout layout, CopyMethod method)`

    * `node_name` – the name of the input node to be fed with external data.
    * `p_data` – the pointer to the first element of external data array.
    * `array_size` – the number of data elements of the external data array.
    * `layout` – the enumerate to specify whether the memory layout is transposed, the available options are:
        · `Predictor::RowMajor` – to hold the original memory sequence of the external data source.
        · `Predictor::ColumnMajor` – to perform matrix transpose while set data to internal container.
    * `method` – the enumerate to specify the copy method to the internal container:
        · `Predictor::Simple` – to copy/cast the arrays element-wise through C++ loop (possibly cache miss).
        · `Predictor::Eigen` – to copy/cast the arrays via `Eigen` library (SIMD is enabled).

  To feed the internal input data container registered under `node_name` with external data source specified by the pointer to the array and the number of data elements. If the data types of the internal container and external source are different, this function would automatically cast the datatype to resolve the difference. The memory layout would be changed during the mapping process if `Predictor::ColumnMajor` is passed to the function. The copy/cast procedure could be accelerated via `Eigen` library by specifying `Predictor::Eigen` options.

- **C05: Run Model:**
  - `run()`

    To run the model prediction, the result will be stored in the internal data container holding the model's output.

- **C06: Get Output Data:**
  - `get_node_data(std::string node_name, std::vector<T>& data)`
    * `node_name` – the name of the output node's data to be extracted to external array.
    * `data` – the external data defined with C++ STL library (i.e., `std::vector`)

    To extract the data stored in the the internal output data container registered under `node_name` into external data array hold by a C++ standard template library (STL) `std::vector`. If the data types of the internal container and external source are different, this function would automatically cast the datatype to resolve the difference.
  - `get_node_data(std::string node_name, std::vector<T>& data, DataLayout layout)`
    * `node_name` – the name of the output node's data to be extracted to external array.
    * `data` – the external data defined with C++ STL library (i.e., `std::vector`)
    * `layout` – the enumerate to specify whether the memory layout is transposed, the available options are:
        · `Predictor::RowMajor` – to hold the original memory sequence of the external data source.
        · `Predictor::ColumnMajor` – to perform matrix transpose while set data to internal container.

    To extract the data stored in the the internal output data container registered under `node_name` into external data array hold by a C++ standard template library (STL) `std::vector`. If the data types of the internal container and external source are different, this function would automatically cast the datatype to resolve the difference. The memory layout would be changed during the mapping process if `Predictor::ColumnMajor` is passed to the function.
  - `get_node_data(std::string node_name, std::vector<T>& data, DataLayout layout, CopyMethod method)`
    * `node_name` – the name of the output node's data to be extracted to external array.
    * `data` – the external data defined with C++ STL library (i.e., `std::vector`)
    * `layout` – the enumerate to specify whether the memory layout is transposed, the available options are:
        · `Predictor::RowMajor` – to hold the original memory sequence of the external data source.
        · `Predictor::ColumnMajor` – to perform matrix transpose while set data to internal container.
    * `method` – the enumerate to specify the copy method to the internal container:
        · `Predictor::Simple` – to copy/cast the arrays element-wise through C++ loop (possibly cache miss).
        · `Predictor::Eigen` – to copy/cast the arrays via `Eigen` library (SIMD is enabled).

    To extract the data stored in the the internal output data container registered under `node_name` into external data array hold by a C++ standard template library (STL) `std::vector`. If the data type of the internal container and external source are different, this function would automatically cast the datatype to resolve the difference. The memory layout would be changed during the mapping process if `Predictor::ColumnMajor` is passed to the function. The copy/cast procedure could be accelerated via `Eigen` library by specifying `Predictor::Eigen` options.

- `get_node_data(std::string node_name, T* p_data, int array_size)`
  - `node_name` – the name of the output node's data to be extracted to external array.
  - `p_data` – the pointer to the first element of external data array.
  - `array_size` – the number of data elements of the external data array.

  To extract the data stored in the the internal output data container registered under `node_name` into external data array specified by the pointer to the array and the number of data elements. If the data type of the internal container and external source are different, this function would automatically cast the datatype to resolve the difference.
- `get_node_data(std::string node_name, T* p_data, int array_size, DataLayout layout)`
  - `node_name` – the name of the output node's data to be extracted to external array.
  - `p_data` – the pointer to the first element of external data array.
  - `array_size` – the number of data elements of the external data array.
  - `layout` – the enumerate to specify whether the memory layout is transposed, the available options are:
    - `Predictor::RowMajor` – to hold the original memory sequence of the external data source.
    - `Predictor::ColumnMajor` – to perform matrix transpose while set data to internal container.

  To extract the data stored in the the internal output data container registered under `node_name` into external data array specified by the pointer to the array and the number of data elements. If the data type of the internal container and external source are different, this function would automatically cast the datatype to resolve the difference. The memory layout would be changed during the mapping process if `Predictor::ColumnMajor` is passed to the function.
- `get_node_data(std::string node_name, T* p_data, int array_size, DataLayout layout, CopyMethod method)`
  - `node_name` – the name of the output node's data to be extracted to external array.
  - `p_data` – the pointer to the first element of external data array.
  - `array_size` – the number of data elements of the external data array.
  - `layout` – the enumerate to specify whether the memory layout is transposed, the available options are:
    - `Predictor::RowMajor` – to hold the original memory sequence of the external data source.
    - `Predictor::ColumnMajor` – to perform matrix transpose while set data to internal container.
  - `method` – the enumerate to specify the copy method to the internal container:
    - `Predictor::Simple` – to copy/cast the arrays element-wise through C++ loop (possibly cache miss).
    - `Predictor::Eigen` – to copy/cast the arrays via `Eigen` library (SIMD is enabled).

  To extract the data stored in the the internal output data container registered under `node_name` into external data array specified by the pointer to the array and the number of data elements. If the data type of the internal container and external source are different, this function would automatically cast the datatype to resolve the difference. The memory layout would be changed during the mapping process if `Predictor::ColumnMajor` is passed to the function. The copy/cast procedure could be accelerated via `Eigen` library by specifying `Predictor::Eigen` options.

- **Auxiliary Functions:**
  - `print_operations()`

    To print all the node information (type and shape) in the loaded model.
  - `print_operations(std::string node_name)`
    - `node_name` – the name of node to print shape and type information.

    To print the node information (type and shape) with specified name in the loaded model.