Infero

ECMWF

Aug 11, 2023

USER GUIDE:

Warning: This software is still under development and not yet ready for operational use.

CHAPTER

ONE

DESCRIPTION

Infero is a machine learning support library that runs a pre-trained machine learning model for inference. Infero provides a common interface to multiple inference engines and can be called from multiple languages (C/C++, Fortran, Python).

Infero requires a relatively small dependency stack and is therefore also suitable for tightly-managed HPC environments.

1.1 Quick Start

Infero can be easily installed and run in few simple steps:

```
# Download Infero
cd $HOME
git clone https://github.com/ecmwf-projects/infero.git

# Install Infero and all its dependencies
cd infero/dev && ./1_install_deps.sh && ./2_install_infero.sh

# Run a ONNX model for inference from a C++ example application
$HOME/builds/infero/bin/2_example_mimo_cpp $HOME/infero/tests/data/mimo_model/mimo_model.
↪onnx onnx input_1 input_2 dense_6
```

On successful run, Infero reports an output similar to the one shown here:

```
Checking key path
Checking key type

ONNX model has: 2 inputs
Layer [0] input_1 has shape: -1, 32,
Layer [1] input_2 has shape: -1, 128,
ONNX model has: 1 outputs
Layer [0] dense_6 has shape: -1, 1,

doing inference..
Tensor(right=0,shape=[3,1,],array=[168172,336345,504516,])

========== Infero Model Statistics ==========
INFERO-STATS: Time to copy/reorder Input  : 2e-06 second (3e-06 second CPU). Updates: 1
INFERO-STATS: Time to execute inference   : 0.000171 second (0.000172 second CPU).␣
↪Updates: 1
```

(continues on next page)

(continued from previous page)

```
INFERO-STATS: Time to copy/reorder Output : 1e-06 second (2e-06 second CPU). Updates: 1
INFERO-STATS: Total Time                  : 0.000174 second (0.000177 second CPU).␣
↪Updates: 3
```

For more information on this and other available examples, see Section *Examples*.

1.2 Build & Install

1.2.1 Build dependencies

Compilation dependencies

- C/C++ compiler
- Fortran 90 compiler
- CMake > 3.16
- ecbuild — ECMWF library of CMake macros

Runtime dependencies:

- eckit — ECMWF C++ toolkit

Optional runtime dependencies:

- TensorFlow Lite
- TensorFlow C-API
- ONNX-Runtime
- TensorRT

1.2.2 Installation scripts

Utility installation scripts are provided in the /dev directory and can be used for default installation of Infero.

- env.sh : defines installation environment
- 1_install_deps.sh : installs dependencies
- 2_install_infero.sh : installs Infero

Installation environment can also be customised by editing the following variables in the *env.sh* script:

Variable	Description	Default
INFERO_VERBOSE_COMPILATION	Verbose flag	0
ROOT_DIR	Infero root path	${HOME}
ROOT_SRC_DIR	Sources root path	${ROOT_DIR}/local
ROOT_BUILD_DIR	build root path	${ROOT_DIR}/builds
ROOT_INSTALL_DIR	install root path	${ROOT_DIR}/installs
WITH_MPI	Use MPI functionalities	OFF
WITH_FCKIT	Use FCKIT (Fortran API)	ON
WITH_ONNX_RUNTIME	ONNX runtime	ON
WITH_TFC_RUNTIME	Tensorflow C-API	ON
TFC_GPU	TEnsorflow C-API (GPU)	1
WITH_TFLITE_RUNTIME	TensorFlow TFlite	OFF
WITH_TRT	TensorRT	OFF
ENABLE_TESTS	Build Infero tests	ON
BUILD_NPROCS	Num procs for building	8

1.2.3 Manual Installation

This installation procedure gives more control on the building/installation process. Infero employs an out-of-source build/install based on CMake. To manually invoke cmake, make sure that ecbuild is installed and the ecbuild executable script is found.

```
which ecbuild
```

Now proceed with installation as follows:

```
# Environment --- Edit as needed
srcdir=$(pwd)
builddir=build
installdir=$HOME/local
```

Create the build directory:

```
mkdir $builddir
cd $builddir
```

Run CMake:

```
ecbuild --prefix=$installdir -- -DECKIT_PATH=<path/to/eckit/install> $srcdir
```

Compile and Install:

```
make -j10
make install
```

Useful Cmake arguments:

Variable	Description
-DENABLE_TESTS	Enable Infero tests
-DCMAKE_INSTALL_PREFIX	Installation root path
-DCMAKE_Fortran_MODULE_DIRECTORY	Fortran module path
-Deckit_ROOT	eckit root path
-DENABLE_MPI	Enable MPI
-DENABLE_FCKIT	Enable fckit
-DFCKIT_ROOT	fckit root path
-DENABLE_TF_LITE	Enable Tensorflow lite
-DTENSORFLOWLITE_PATH	TensorFlow lite sources path
-DTENSORFLOWLITE_ROOT	TensorFlow lite root path
-DENABLE_TF_C	Enable TensorFlow C-API
-DTENSORFLOWC_ROOT	TensorFlow C-API root path
-DENABLE_ONNX	Enable onnx-runtime
-DONNX_ROOT	ONNX-runtime root path
-DENABLE_TENSORRT	Enable tensor-rt
-DTENSORRT_ROOT	TensorRT root path

1.2.4 Run Tests

Tests can be run from the script:

```
dev/3_run_tests.sh
```

Note: The following environment variables can also be set when running tests:

- *INFERO_TEST_NPROCS*: number of processors to use for each regression test (when MPI is enabled)
- *INFERO_TEST_TOL*: overrides the error tolerance on tests at runtime

1.3 Examples

1.3.1 Overview

Infero comes with examples that demonstrate how to use Infero API's for different languages. Examples can be found in:

- <infero-source-path>/examples

And when compiled, the corresponding executables are found in:

- <infero-build-path>/bin

The examples show how to use Infero for a multi-input single-output Machine Learning model from C, C++ and Fortran (same model and input data are used for all the cases, so also same output is expected).

- *1_example_mimo_c.c*
- *2_example_mimo_cpp.cc*
- *3_example_mimo_fortran.F90*
- *4_example_mimo_thread.cc*

1.3.2 Run the examples

The examples can be run as follows (in this specific case shown below, the onnx backend is used - therefore to run the example as below, make sure that ONNX backend is installed - see *Build & Install*).

Note that here below <path/to/mimo/model> is: <path/to/infero/sources>/tests/data/mimo_model

C example:

```
cd <path/to/infero/build>
./bin/1_example_mimo_c <path/to/mimo/model>/mimo_model.onnx onnx input_1 input_2 dense_6
```

C++ example:

```
cd <path/to/infero/build>
./bin/2_example_mimo_cpp <path/to/mimo/model>/mimo_model.onnx onnx input_1 input_2 dense_
↪6
```

Fortran example:

```
cd <path/to/infero/build>
./bin/3_example_mimo_fortran <path/to/mimo/model>/mimo_model.onnx onnx input_1 input_2␣
↪dense_6
```

C++ threaded example:

```
cd <path/to/infero/build>
./bin/4_example_mimo_thread <path/to/mimo/model>/mimo_model.onnx onnx input_1 input_2␣
↪dense_6
```

1.3.3 Code Explained

The examples are extensively commented to describe the usage of the API's step-by-step. A brief description of the main sections from the Fortran example is also reported here below (for the full example, refer to *3_example_mimo_fortran.F90*).

This section below contains the declaration of the necessary input variables. t1 and t2 are the fortran arrays containing input data and t1_name and t2_name are the names of the input layers to which the tensors will be assigned.

```
! input tensors
real(c_float) :: t1(n_batch,32) = 0
real(c_float) :: t2(n_batch,128) = 0

! names of input layers
character(len=128) :: t1_name
character(len=128) :: t2_name
```

The association between tensors and names of the corresponding input layers is then made through a key/value container of type *fckit_map* (here below the necessary declarations):

```
! auxiliary fckit tensor wrappers
type(fckit_tensor_real32) :: tensor1
type(fckit_tensor_real32) :: tensor2
```

(continues on next page)

(continued from previous page)

```
! key/value map for name->tensor
type(fckit_map) :: imap
```

Output tensor(s) are declared and arranged into an *fckit_map* in the same way.

```
! output tensor
real(c_float) :: t3(n_batch,1) = 0

! name of output layer
character(len=128) :: t3_name

! auxiliary fckit tensor wrappers
type(fckit_tensor_real32) :: tensor3

! key/value map for name->tensor
type(fckit_map) :: omap
```

The type for the machine learning model is called *infero_model*:

```
! the infero model
type(infero_model) :: model
```

Input tensors are filled row-wise with dummy values for this example and the *fckit_map* is filled in:

```
! fill-in the input tensors
! Note: dummy values for this example!
t1(1,:) = 0.1
t1(2,:) = 0.2
t1(3,:) = 0.3

t2(1,:) = 33.0
t2(2,:) = 66.0
t2(3,:) = 99.0

! init infero library
call infero_check(infero_initialise())

! wrap input tensors into fckit_tensors
tensor1 = fckit_tensor_real32(t1)
tensor2 = fckit_tensor_real32(t2)

! construct the fckit input map
imap = fckit_map()

! insert entries name+tensor into the input map
call imap%insert(TRIM(t1_name), tensor1%c_ptr())
call imap%insert(TRIM(t2_name), tensor2%c_ptr())
```

Same thing is done for the output tensor

```
! wrap output tensor into fckit_tensor
tensor3 = fckit_tensor_real32(t3)
```

(continues on next page)

(continued from previous page)

```
! construct the fckit output map
omap = fckit_map()

! insert entry name+tensor into the output map
call omap%insert(TRIM(t3_name), tensor3%c_ptr())
```

Configure and call infero inference method

```
! YAML configuration string string
yaml_config = "---"//NEW_LINE('A') &
  //"  path: "//TRIM(model_path)//NEW_LINE('A') &
  //"  type: "//TRIM(model_type)//c_null_char

! get a inference model model
call infero_check(model%initialise_from_yaml_string(yaml_config))

! run inference
call infero_check(model%infer(imap, omap))
```

Print inference statistics, configuration and output values

```
! explicitely request to print stats and config
call infero_check(model%print_statistics())
call infero_check(model%print_config())

! print output
call infero_check(oset%print())
```

Finally free the allocated memory for the input and output tensor sets and, free the model and finalise the library itself

```
! free the model
call infero_check(model%free())

! finalise fckit objects
call tensor1%final()
call tensor2%final()
call tensor3%final()
call imap%final()
call omap%final()

! finalise library
call infero_check(infero_finalise())
```

CHAPTER

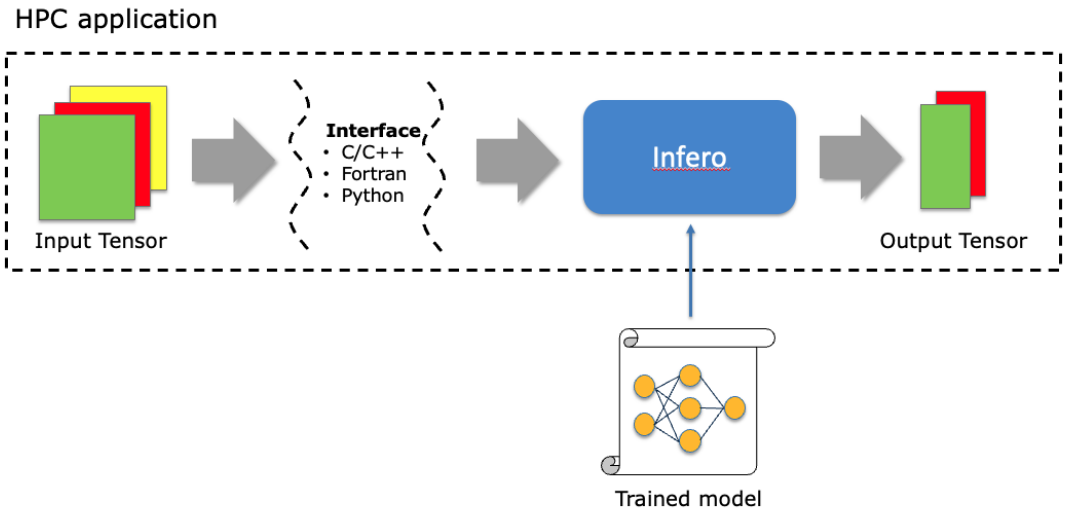
TWO

ARCHITECTURE

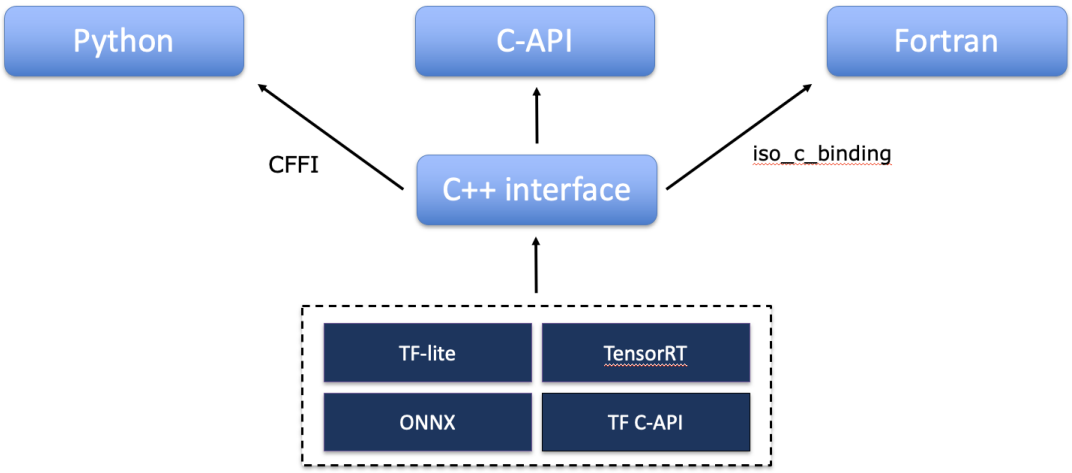
Infero is designed as a thin layer on top of interchangeable inference engines (backends) and provides a common API usable by multiple programming languages. It performs three main steps:

1. Transfers input data from the caller application to the inference engine
2. Runs the inference engine
3. Transfers output data back to the caller application

Infero accepts commonly used input/output data formats for each language of the API. For example, a C++ application will transfer data to/from Infero as raw memory buffers, a Python application will use numpy objects and a Fortran application Fortran arrays.



Infero is primarily developed in C++ and the additional APIs for C, Fortran and Python are built on top of the C++ code. A schematic diagram of the API architecture is shown here below:


Python
C-API
Fortran
CFFI
C++ interface
iso_c_binding
TF-lite
TensorRT
ONNX
TF C-API

CHAPTER

THREE

FEATURES

- **API available for multiple languages:**
  - C, C++, Fortran, Python
- **Inference Engines supported:**
  - TensorFlow LITE
  - TensorFlow C-API
  - ONNX-Runtime
  - TensorRT
- Support for Multiple-input Multiple-output models
- Automatic handling of C-style and Fortran-style tensors

CHAPTER

FOUR

LICENSE

Infero is available under the open source Apache License Version 2. In applying this licence, ECMWF does not waive the privileges and immunities granted to it by virtue of its status as an intergovernmental organisation nor does it submit to any jurisdiction.

Authors
Antonino Bonanni, James Hawkes, Tiago Quintino

Version
0.1.0