STL/Eigen - Automatic conversion and plugins for Python based ML-backends

Khushiyant

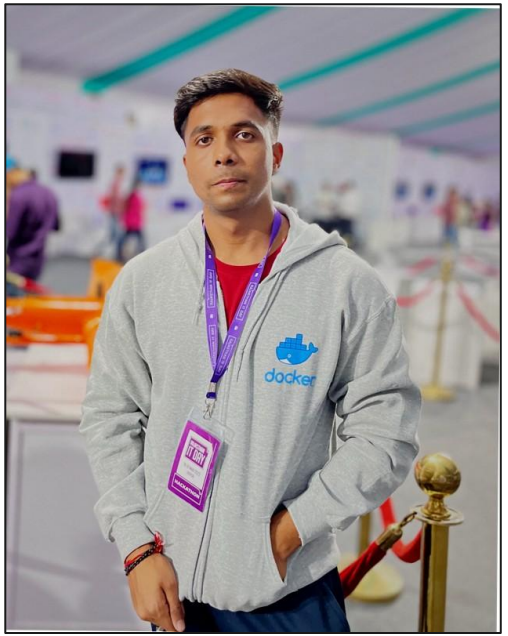
About Me

Name - Khushiyant

Student - B.Tech in Computer Science and Engineering

Professionally - MLE @ Martian

Interest in Tech - Container Orchestration, ML Infra



What is cppyy?

cppyy is a powerful, automatic Python-C++ bindings generator that lets you seamlessly call C++ from Python and vice versa. It offers high performance, lazy loading for large projects, cross-inheritance, and interactive exploration of C++ libraries—all without the need for intermediate languages or tedious boilerplate code. Imagine dynamically mixing Python and C++ features with ease, thanks to cppyy's use of Cling, the C++ interpreter, which matches Python's dynamism and interactivity. cppyy is future-proof, supporting advanced C++ features and modern compilers, effortlessly handling complex tasks like working with Boost's boost::any. cppyy can achieve near C++ performance, making it ideal for large-scale, distributed development environments.

Example

```
>>> cppyy.cppdef(r"""\
... void hello() {
...     std::cout << "Hello, World!" << std::endl;
... }""")
True
>>> cppyy.gbl.hello()
Hello, World!
```

Problem Statement

Current support follows container types in STL like **std::vector**, **std::map**, and **std::tuple** and the Matrix-based classes in Eigen/Dense. Presently, Cpppy's stl::vector is accessed by cpppy.gbl.std.vector doesn't support arbitrary dimensions, and there is no support for conversion mechanisms between Python built-in types, numpy.ndarray, and STL/Eigen data structures.

What are we trying to achieve hopefully

Cppyy for CPU operations in ML backends

Arbitrary Dimension Support for STL Vector

Our major goal is to facilitate CPU operations for machine learning backends (such as JAX), however the lack of support for arbitrary dimensions is the first impediment.

Also, Provide support for basic rudimentary types that could be essential in further conversion utilities later.

For example, arbitary dimension support will allow processing types like ***<class cppyy.gbl.vector<cppyy.gbl.std.vector<double>> at 0x121053940>***

Conversion Mechanism

Since, our goal is to provide support for JAX CPU operations, it becomes kind of mandatory due to current overhead (2.2x) such as in this case of eigen comma insertions.

We are planning to implement better initialisation as well as conversion utilities to facilitate the implementation for JAX plugins

Cppyy Eigen multiplication with Comma insertion	JAX without GPU using lax.dot
No GPU/TPU found, falling back to CPU. (Set TF_CPP_MIN_LOG_LEVEL=0 and rerun for more info.) Running tests under Python 3.8.10: /usr/bin/python [RUN] AnnTest.test_approx_max_k0 (qy_shape=(200, 128), db_shape=(128, 500), dtype=<class 'numpy.float32'>, k=1, recall=0.95) Scores : [[-47.577297 183.75833 -81.97412 ... -50.89401 -29.767101 68.44914] [OK] Ran 20 tests in 3.397s	No GPU/TPU found, falling back to CPU. (Set TF_CPP_MIN_LOG_LEVEL=0 and rerun for more info.) Running tests under Python 3.8.10: /usr/bin/python [RUN] AnnTest.test_approx_max_k0 (qy_shape=(200, 128), db_shape=(128, 500), dtype=<class 'numpy.float32'>, k=1, recall=0.95) Scores : [[-47.5773 183.75835 -81.97411 ... -50.894028 -29.767109 68.44914] [OK] Ran 20 tests in 1.517s

JAX Plugins and benchmarking

After completion of above milestones, we will implement the JAX plugins where we try to replace JAX implementation with cppyy for CPU computes such as matrics, sparse based calculations. Later, it is planned to undergo the benchmarking to facilitate and quantify the comparison between JAX and our's implementation.

Q&A