# Adopting CppInterOp in cppyy

## Contact Details

Name: Vipul Cariappa
Email: vipulcariappa@gmail.com
GitHub: https://github.com/Vipul-Cariappa/
Website: https://www.vipulcariappa.xyz/
Resume: https://www.vipulcariappa.xyz/assets/My-Resume.pdf
Timezone: India Standard Time (GMT+5:30)

## About My Project Proposal

cppyy is a Python library that provides fully automatic, dynamic Python-C++ bindings using the cling based C++ interpreter/incremental compiler. Cling itself is based on the LLVM toolchain. Compiler Research maintains its own fork of cppyy. The main motivation of this fork is to directly use the LLVM's clang-REPL for C/C++ interpreter/incremental compiler. This reduces the burden on the developers to maintain a separate fork of LLVM like the cling interpreter. The Compiler Research's cppyy fork's migration to using the upstream clang-REPL is an ongoing effort. The migration is incomplete, and many tests fail in the Compiler Research's fork. I aim to fix as many failing tests as possible to complete the transition to using the clang-REPL compiler backend.
Once we get most of the tests to pass using the clang-REPL backend, I plan on implementing a multi-language jupyter kernel on top of xeus-cpp. This multi-language kernel will support using both C++ and Python for code execution and will provide the ability for the users to seamlessly use symbols defined in each other.

I plan on devoting the first 8 to 10 weeks to fixing the failing tests at cppyy on the clang-REPL backend. And work on the multi-language kernel for the following 2 to 4 weeks.

## Benefit to the Community

If we complete the migration to clang-REPL we will not be required to maintain a separate LLVM's fork, which we do through cling. We can directly use the upstream LLVM, and use all of its new features, speed improvements, and bug fixes. The current cppyy uses lots of string manipulation. This string manipulation increases the performance overhead and affects the code readability and debuggability. While using clang-REPL, these string manipulations can be avoided, mitigating the drawbacks mentioned.
Python is the go-to language for data science and machine learning. But Python is slow because it is interpreted and dynamically typed. Whereas C++ is compiled and statically typed, C++ offers much better performance compared to Python. Most of the compute intensive Python

libraries are written in C/C++ or Fortran. Building a multi-language Jupyter kernel will provide users with the advantages of both worlds. Easy to write Python, and performance of C++.

## Tentative Timeline

- Week 1 & 2: Fix failing tests in `test_operators.py`. Going through the failing tests in this file, I could easily understand the requirements to fix most of the failing tests. It would be a good starting point.
- Week 3: Fix failing tests in `test_pythonify.py`. The failing tests are related to default arguments and keyword arguments.
- Week 4: Fix failing tests in `test_conversions.py`. These failing tests are related to implicit type conversions. Like C++ iterable to Python iterable and Python `tuple` to C++ `std::pair`.
- Week 5 & 6: Fix failing tests in `test_advancedcpp.py`.
- Week 7 & 8: Fix failing tests related to `stltypes`.
- Week 9 & 10: Fix failing tests related to `template`s , and template initializations.
- Week 11 & 12: Work on xeus-cpp, implementing the multi-language kernel.

xeus-cpp is a Jupyter Kernel for C++. It uses CppInterOp library for incremental compilation and execution of code. The idea behind the multi-language kernel is to use both Python and C++ in a single notebook. The end user should be able to use symbols across languages seamlessly. The kernel would be responsible for any necessary type conversions.
Cppyy library can be used for this purpose. It already has the necessary building blocks for incremental compilation and interoperability with Python. We would adapt xeus-cpp to integrate with cppyy for the Python code execution.

It is difficult to estimate the time requirements for fixing bugs. Therefore the above mentioned timeline should be considered as an estimate.

## Related Work

- Fixed an issue in CppInterOp related to a missing feature; to get the include paths; through this PR.

## Biographical Information

I am currently pursuing my bachelor's degree in computer science and engineering in Bangalore, India. I am a two time GSoC contributor. In 2023, I contributed to GNU Octave. Octave is a programming language designed for compute intensive and scientific applications. My project was on interoperability between Octave and Python programming languages. In 2024, I contributed to LPython under the Python Software Foundation. LPython is a statically typed, compiled version of Python. I built a REPL (read-evaluate-print-loop) shell for LPython, Jupyter Kernel, and worked in interoperability with CPython.