Bridging C++ and Python for BioDynaMo Agent-Based Simulations

Anton Dekov

About me

My name is Anton Dekov and I am a highschool student at the Mathematics high school in Plovdiv Bulgaria.
I have around 4-5 years of experience with C++ and some experience with Python from personal projects.
Through my internship I aim to get practical experience and what working in this field would actually look like.

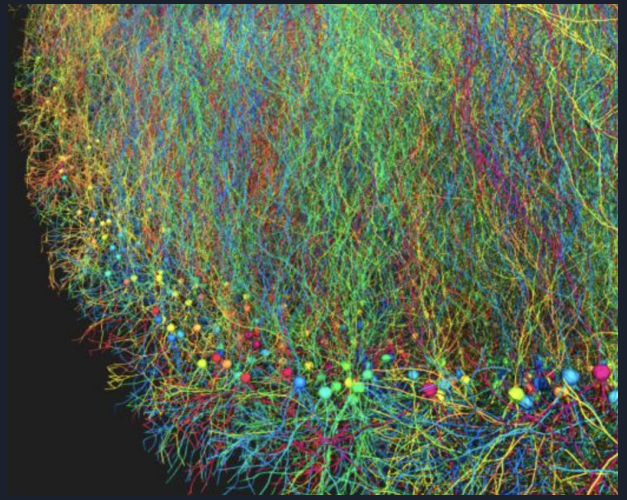
- Personal email: dexant3500@gmail.com
- School email: antondekov_20a@scholmath.eu
- github: Dexant3500

About BioDynaMo

BioDynaMo is a high-performance agent-based simulation platform, capable of simulating up to 1.7 billion agents on a single server.
It's is developed by CERN openlab and Newcastle University.

BioDynaMo can be used for:

- Cancer research and radiotherapy simulations
- Epidemiology and disease spread simulations
- Neuroscience and neuronal growth simulations
- Social science modeling

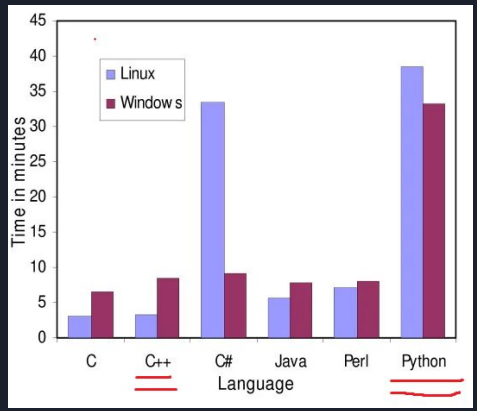


BIODYNAMO
BIOLOGY DYNAMICS MODELLER

The problem

BioDynaMo's performance-critical core is written in C++, which is excellent for speed but creates significant barriers for many users. Currently, creating new agent types requires writing C++ code from scratch, which makes it harder for non-C++ users and students who are still learning to adopt it. It also means every change demands recompilation slowing down progress.

There is also an ecosystem gap. Python has many helpful libraries especially for data analysis which are currently not easily accessible for BioDynaMo users.



The solution

Introducing cppyy

Cppyy is an automatic, run-time Python-C++ bindings generator developed at Lawrence Berkeley National Laboratory. Based on Cling (a C++ interpreter built on Clang/LLVM), it requires no manual binding code and generates bindings on-the-fly.
What makes cppyy suited for this project is its support for cross-inheritance - Python classes can inherit directly from C++ base classes. This enables true hybrid development where performance-critical infrastructure stays in C++ while agent behaviors are defined in Python, with near-native performance when using PyPy.

```
import cppyy

cppyy.cppdef(
    """
class MyClass {
private:
    int myInt;
public:
    MyClass(int myInt) {
        this->myInt = myInt;
    }
    int getMyInt() {
        return myInt;
    }
};
    """
)

from cppyy.gbl import MyClass
print("Enter ClassID")
ClassID=int(input())
my_var = MyClass(ClassID)
print(f"Class ID recorded as: {my_var.getMyInt()}")
```

```
Enter ClassID
27
Class ID recorded as: 27
```

(Simple example code)

Project goals

My main goal is to enable BioDynaMo users to write simulation agents in Python that seamlessly inherit from and extend existing C++ agent classes.

Below are some of my subgoals along the way:

- Establish the connection between BioDynaMo and cppyy
- Handle BioDynaMo specific types
- Enable cross-language inheritance
- Achieve virtual method overriding
- Integrate with AgentManager

Project plan

- Phase 1: Basic cppyy Integration (Weeks 1-4)

  Set up the environment and establish connectivity. Load a compiled C++ BioDynaMo library in Python, instantiate an existing C++ agent, and call its basic methods like GetPosition(). This proves the fundamental connection works.

- Phase 2: Python Inheritance from C++ (Weeks 5-10)

  Handle BioDynaMo's complex types (Double3, AgentUid, Real3), create a Python class that inherits from a C++ agent, override virtual methods like Run(), and ensure the C++ simulation calls the Python implementations. Integrate with AgentManager for proper lifecycle management.

Project plan

- Phase 3: Documentation (Weeks 11-14)

  Create comprehensive setup guides, Jupyter notebooks with working examples, and clean commented code. Deliver a complete project package with a short demo video and final report - making everything usable for others.

- Phase 4: Final delivery (Week 15)

By the end, my goal is for someone else to be able to follow the documentation to load a C++ agent, create their own Python agent inheriting from it, and override its behavior without needing extensive knowledge of C++.

Thank you for your attention!