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

Mentors: Aaron Jomy, Vassil Vassilev, Wim Lavrijsen, Jonas Rembser

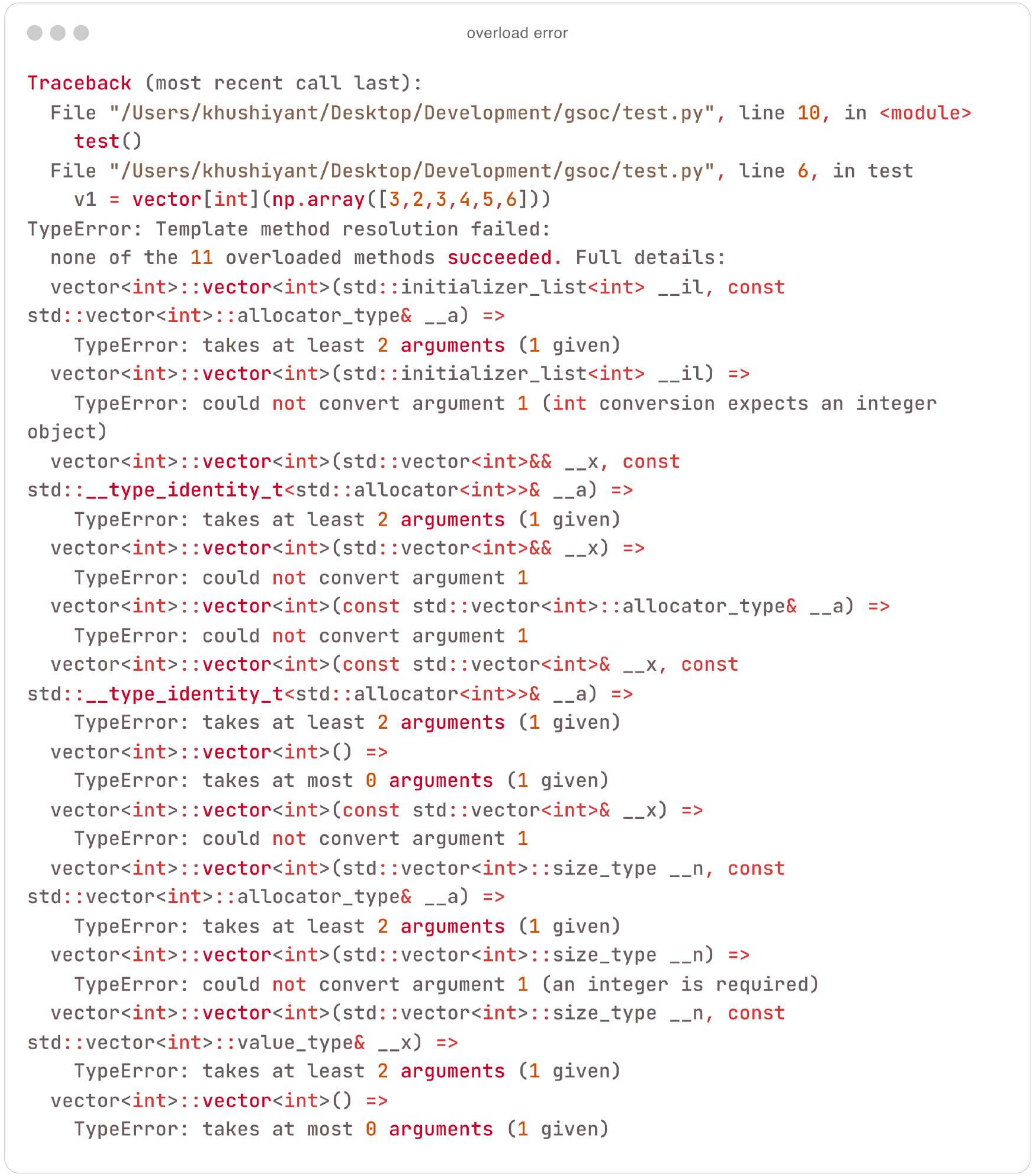
Khushiyant

Problem

- Pythonization of vector initialisation don't handle buffer (i.e. numpy arrays, strings) well

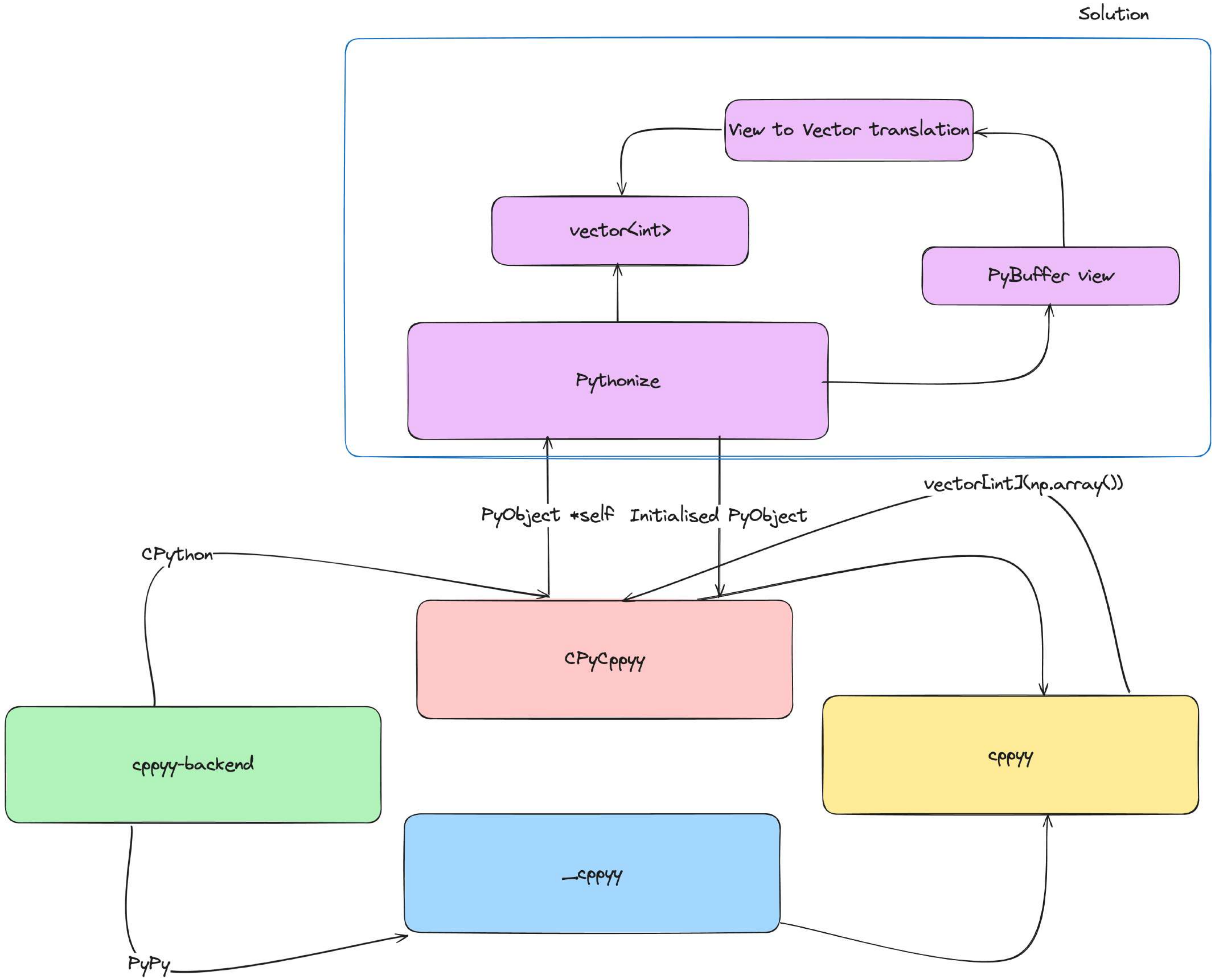
- Leads to unsuccessful overloading of templates

- It requires a separate logic to handle numpy array construction



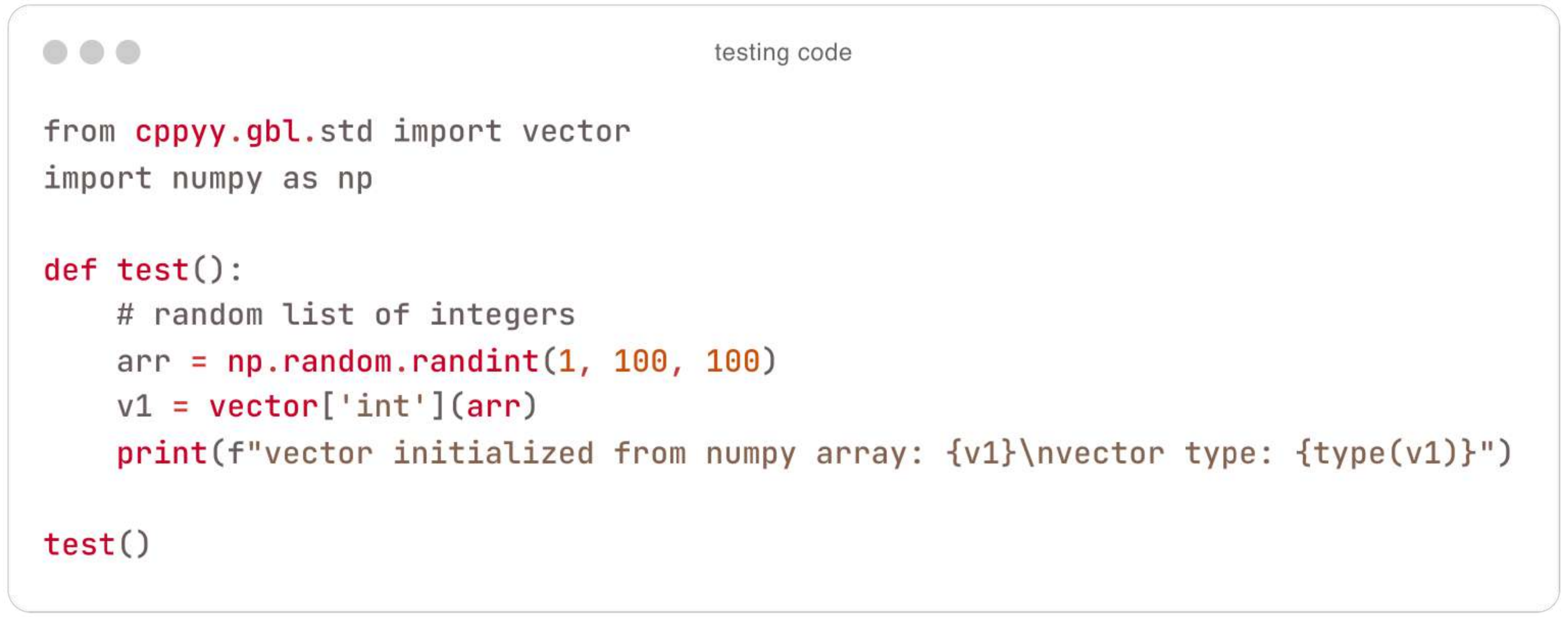
```
Traceback (most recent call last):
  File "/Users/khushiyant/Desktop/Development/gsoc/test.py", line 10, in <module>
    test()
  File "/Users/khushiyant/Desktop/Development/gsoc/test.py", line 6, in test
    v1 = vector[int](np.array([3,2,3,4,5,6]))
TypeError: Template method resolution failed:
  none of the 11 overloaded methods succeeded. Full details:
  vector<int>::vector<int>(std::initializer_list<int> __il, const
std::vector<int>::allocator_type& __a) =>
    TypeError: takes at least 2 arguments (1 given)
  vector<int>::vector<int>(std::initializer_list<int> __il) =>
    TypeError: could not convert argument 1 (int conversion expects an integer
object)
  vector<int>::vector<int>(std::vector<int>&& __x, const
std::__type_identity_t<std::allocator<int>>& __a) =>
    TypeError: takes at least 2 arguments (1 given)
  vector<int>::vector<int>(std::vector<int>&& __x) =>
    TypeError: could not convert argument 1
  vector<int>::vector<int>(const std::vector<int>::allocator_type& __a) =>
    TypeError: could not convert argument 1
  vector<int>::vector<int>(const std::vector<int>& __x, const
std::__type_identity_t<std::allocator<int>>& __a) =>
    TypeError: takes at least 2 arguments (1 given)
  vector<int>::vector<int>() =>
    TypeError: takes at most 0 arguments (1 given)
  vector<int>::vector<int>(const std::vector<int>& __x) =>
    TypeError: could not convert argument 1
  vector<int>::vector<int>(std::vector<int>::size_type __n, const
std::vector<int>::allocator_type& __a) =>
    TypeError: takes at least 2 arguments (1 given)
  vector<int>::vector<int>(std::vector<int>::size_type __n) =>
    TypeError: could not convert argument 1 (an integer is required)
  vector<int>::vector<int>(std::vector<int>::size_type __n, const
std::vector<int>::value_type& __x) =>
    TypeError: takes at least 2 arguments (1 given)
  vector<int>::vector<int>() =>
    TypeError: takes at most 0 arguments (1 given)
```

Introduction

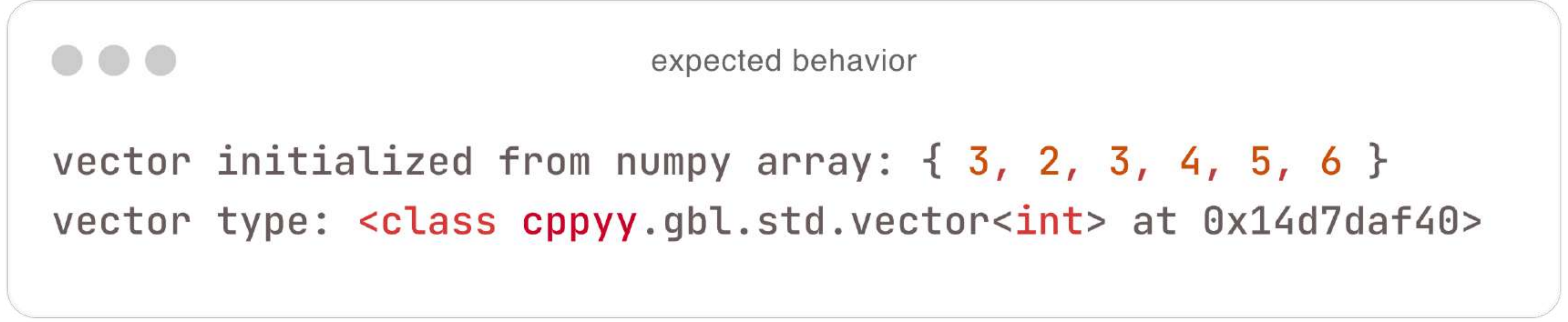


Solution

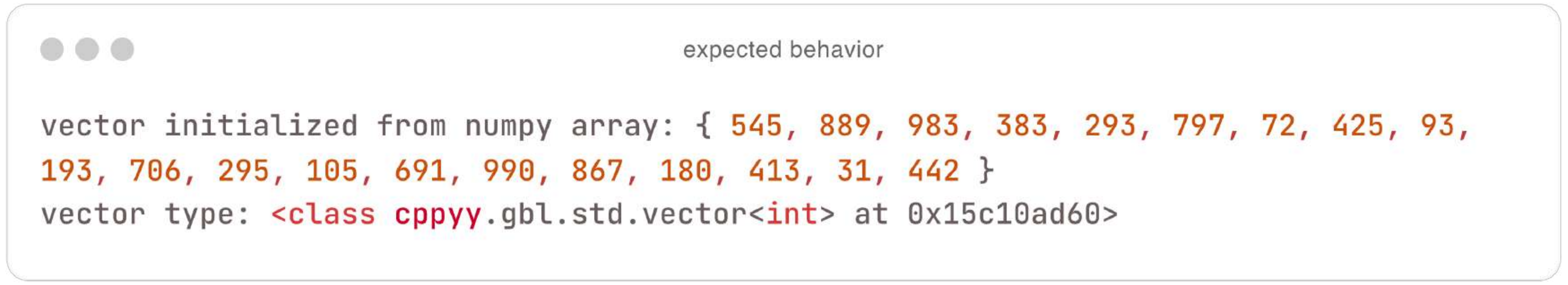
Expectation



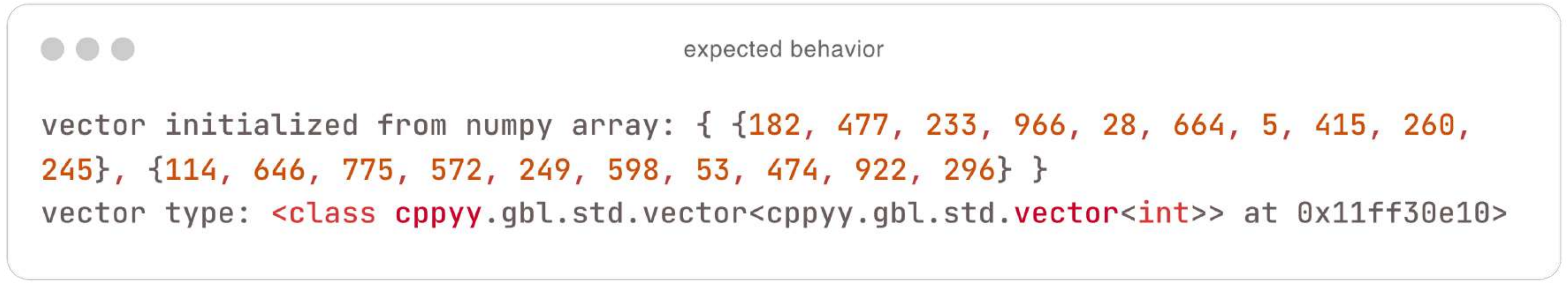
```python
from cppyy.gbl.std import vector
import numpy as np

def test():
    # random list of integers
    arr = np.random.randint(1, 100, 100)
    v1 = vector['int'](arr)
    print(f"vector initialized from numpy array: {v1}\nvector type: {type(v1)}")

test()
```



```
vector initialized from numpy array: { 3, 2, 3, 4, 5, 6 }
vector type: <class cppyy.gbl.std.vector<int> at 0x14d7daf40>
```



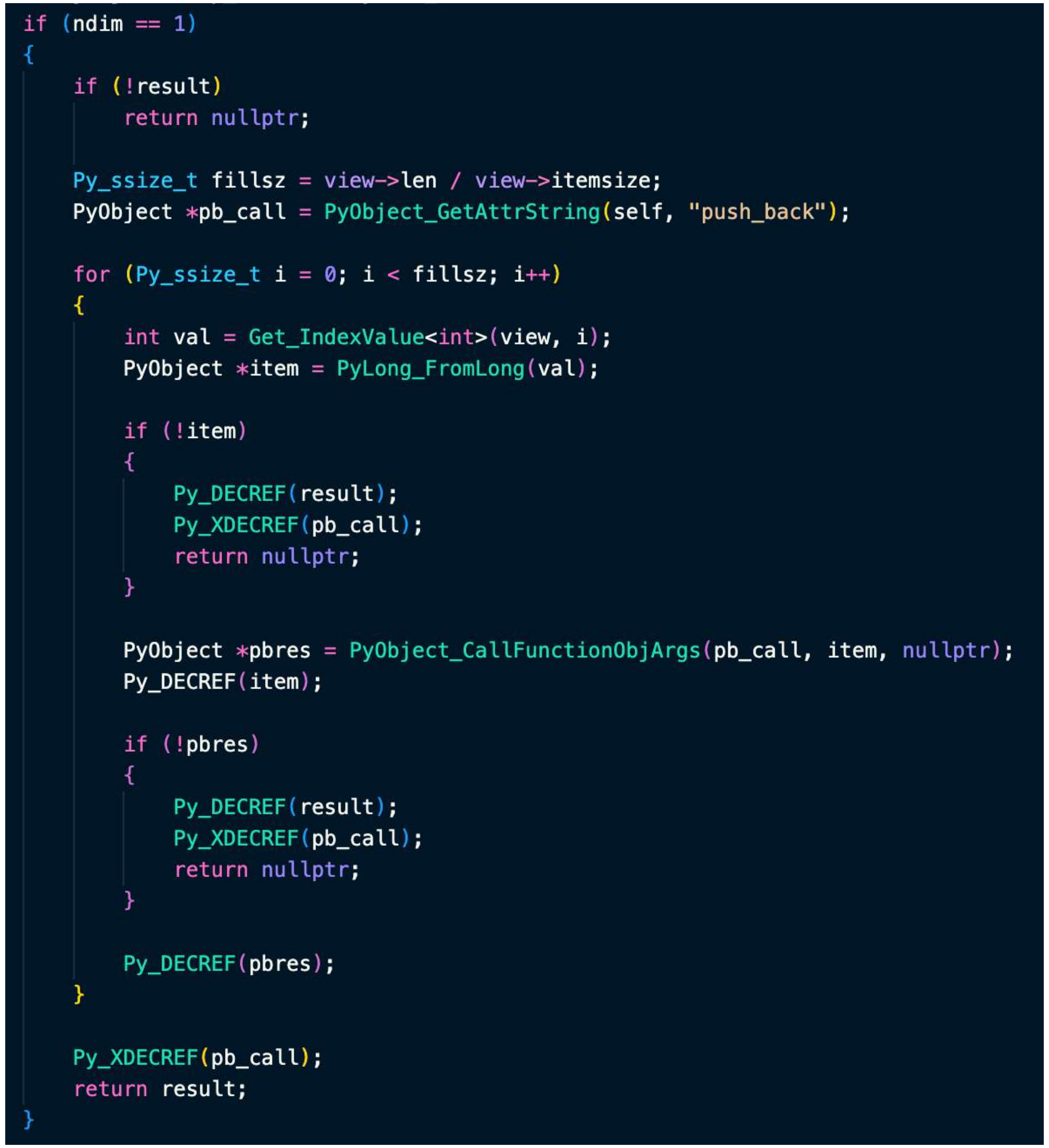
```
vector initialized from numpy array: { 545, 889, 983, 383, 293, 797, 72, 425, 93,
193, 706, 295, 105, 691, 990, 867, 180, 413, 31, 442 }
vector type: <class cppyy.gbl.std.vector<int> at 0x15c10ad60>
```



```
vector initialized from numpy array: { {182, 477, 233, 966, 28, 664, 5, 415, 260,
245}, {114, 646, 775, 572, 249, 598, 53, 474, 922, 296} }
vector type: <class cppyy.gbl.std.vector<cppyy.gbl.std.vector<int>> at 0x11ff30e10>
```

Base Case

- Iterate over the buffer's elements
- Convert each element to a Python long object
- Append each object to the master result object using the push_back method call using python c api's ***PyObject_CallFunctionObjArgs***
- Some other similar functionalities were discarded to support multi-platform support such ***PyList_Append***, ***emplace_back*** as these are can lead to segmentation fault in MacOS memory management system

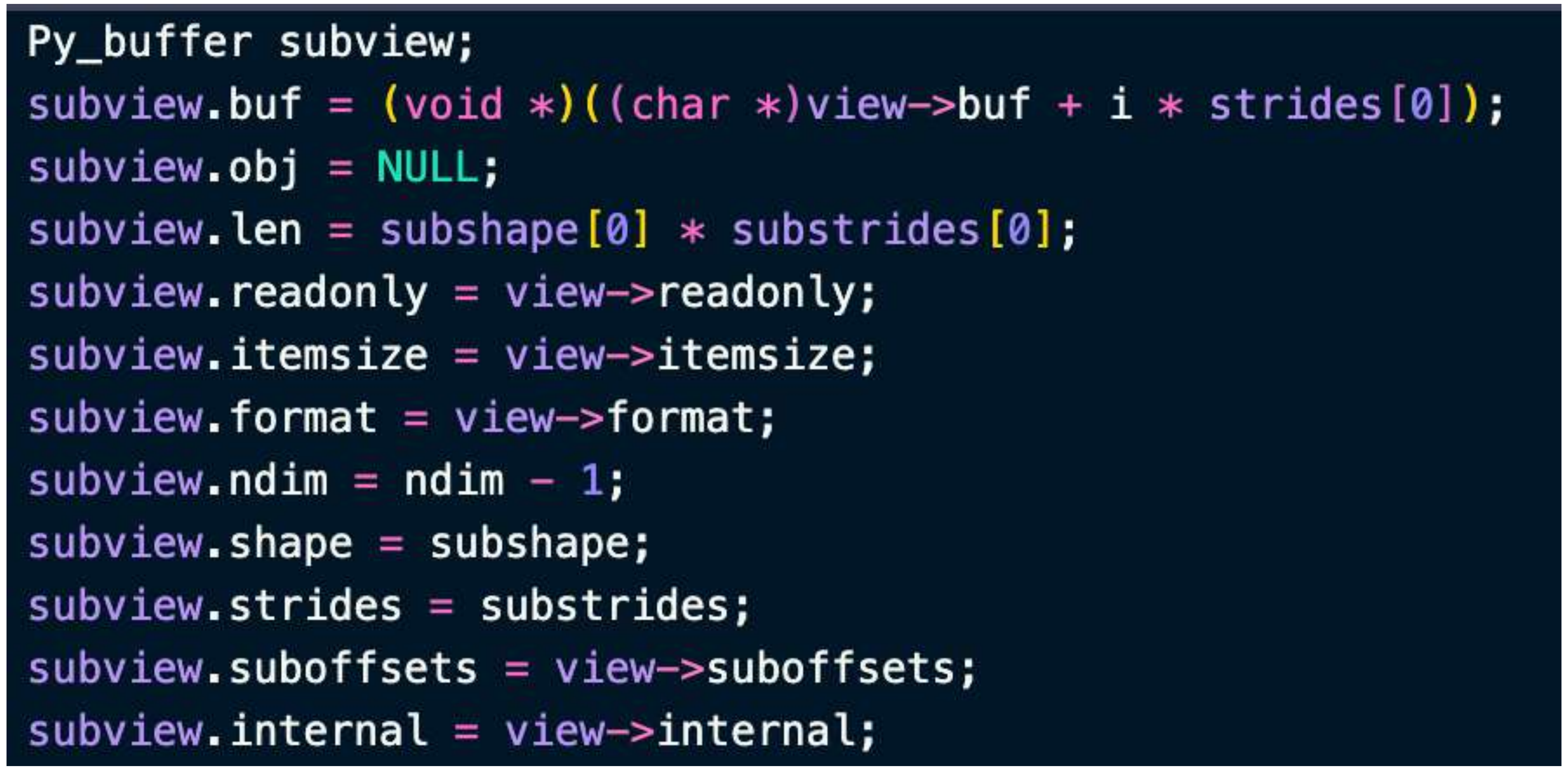


```
if (ndim == 1)
{
    if (!result)
        return nullptr;

    Py_ssize_t fillsz = view->len / view->itemsize;
    PyObject *pb_call = PyObject_GetAttrString(self, "push_back");

    for (Py_ssize_t i = 0; i < fillsz; i++)
    {
        int val = Get_IndexValue<int>(view, i);
        PyObject *item = PyLong_FromLong(val);

        if (!item)
        {
            Py_DECREF(result);
            Py_XDECREF(pb_call);
            return nullptr;
        }

        PyObject *pbres = PyObject_CallFunctionObjArgs(pb_call, item, nullptr);
        Py_DECREF(item);

        if (!pbres)
        {
            Py_DECREF(result);
            Py_XDECREF(pb_call);
            return nullptr;
        }

        Py_DECREF(pbres);
    }

    Py_XDECREF(pb_call);
    return result;
}
```

Intermediate SubView

- Buffer protocol provides a buffer interface that allows you to access the internal data of an object as a contiguous block of memory. It's a way to expose the internal data of an object to other parts of the program, without having to copy the data.

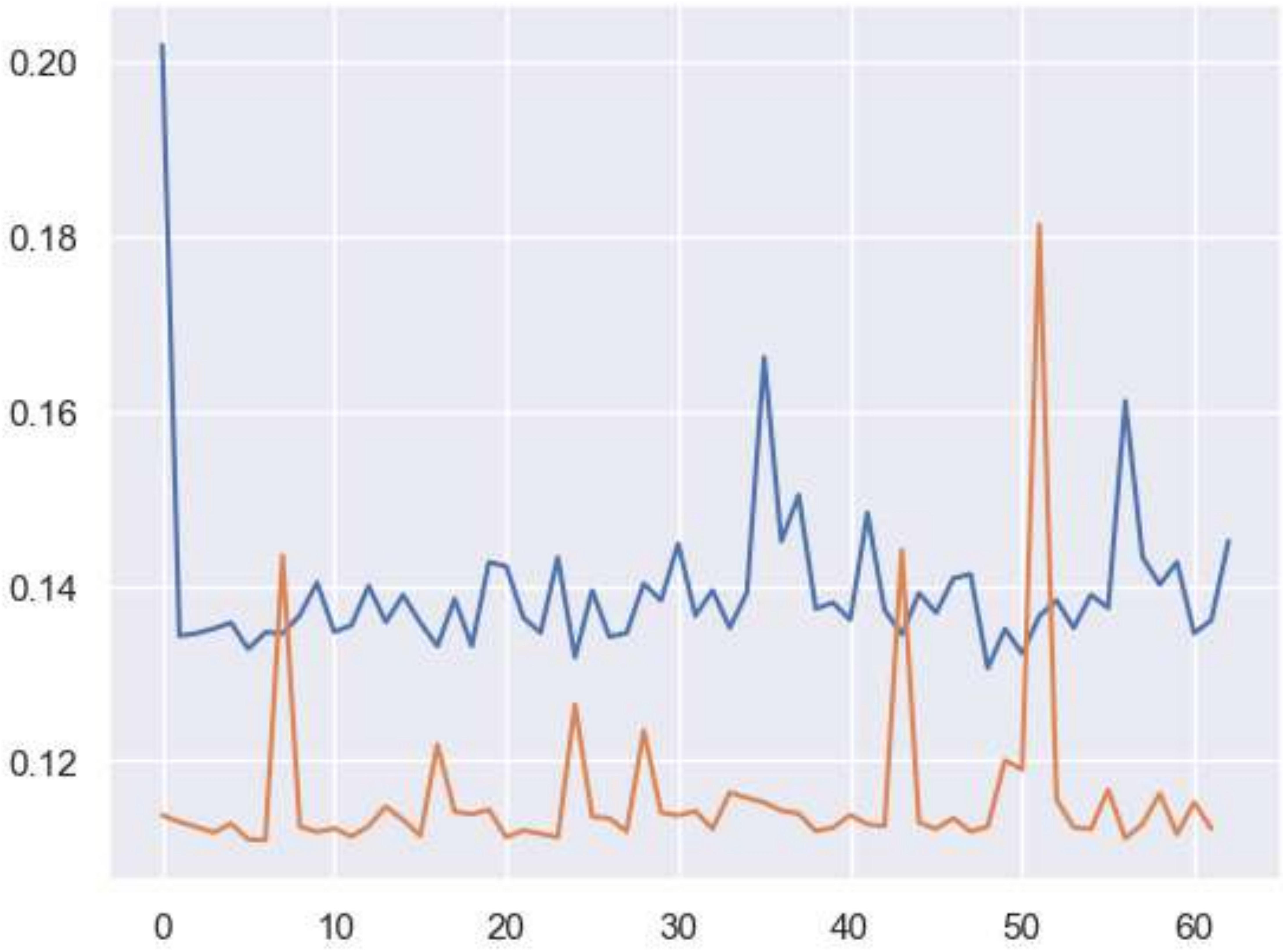
- Buffer view is sliced till the it reaches the ***ndim == 1*** and finally processed independently and added to temporary init objects



```
Py_buffer subview;
subview.buf = (void *)((char *)view->buf + i * strides[0]);
subview.obj = NULL;
subview.len = subshape[0] * substrides[0];
subview.readonly = view->readonly;
subview.itemsize = view->itemsize;
subview.format = view->format;
subview.ndim = ndim - 1;
subview.shape = subshape;
subview.strides = substrides;
subview.suboffsets = view->suboffsets;
subview.internal = view->internal;
```

Results

- Average List Init Time: ***145ms***
- Average Numpy Init Time: ***112ms***
- Numpy has ***~22-28%*** faster init time on average



- *Note: Test included 62 run cycle for vector initialisation from list and numpy array of size with random 1000 samples*

Features Added

- Added support for vector initialisation from numpy array
- Better initialisation time in comparison to list, python array
- Multi platform support - MacOS, Linux
- Useful in planned conversion utilities that are to be used in further ML framework implementations

Further Improvements

Performance Efficiency

- Iteration over a buffer is a expensive operation
- Replace the iteration method with suitable low level utility such as in case of ***VectorIAdd***, we use ***PyStrings:gInsert*** for iteration
- Added multiple memory management checks such as in ***Get_IndexValue***

Further Improvements

Extent Support for Other Datatypes

- Currently, we support lumpy arrays of type ***int*** only
- We are creating run-time datatype mapping utility using low level Convertors and view buffer format

Questions?