My journey of fixing an unmaintained package, spanning Python, C++, nanobind and Github Actions

Gracjan Adamus @ Python Maven

#whoami

- Third-year Applied Computer Science student @ AGH

#whoami

- Third-year Applied Computer Science student @ AGH
- President of KNI Kernel

#whoami

- Third-year Applied Computer Science student @ AGH
- President of KNI Kernel
- 2025 Summer Intern @ Paul Scherrer Institute

#whoami

- Third-year Applied Computer Science student @ AGH
- President of KNI Kernel
- 2025 Summer Intern @ Paul Scherrer Institute
- 2026 Summer Student @ CERN

#whoami

- Third-year Applied Computer Science student @ AGH
- President of KNI Kernel
- 2025 Summer Intern @ Paul Scherrer Institute
- 2026 Summer Student @ CERN
- Previous experience as: C++ Developer

#whoami

- Third-year Applied Computer Science student @ AGH
- President of KNI Kernel
- 2025 Summer Intern @ Paul Scherrer Institute
- 2026 Summer Student @ CERN
- Previous experience as: C++ Developer
- Certified Mole lover

#whoami

- Third-year Applied Computer Science student @ AGH
- President of KNI Kernel
- 2025 Summer Intern @ Paul Scherrer Institute
- 2026 Summer Student @ CERN
- Previous experience as: C++ Developer
- Certified Mole lover
- Interested in HPC, GPUs and all the fun stuff

#whoami

- Third-year Applied Computer Science student @ AGH
- President of KNI Kernel
- 2025 Summer Intern @ Paul Scherrer Institute
- 2026 Summer Student @ CERN
- Previous experience as: C++ Developer
- Certified Mole lover
- Interested in HPC, GPUs and all the fun stuff
- Griger5 @ Github

What is pybind11?

What is pybind11?

- Created by Wenzel Jakob, associate professor at EPFL

What is pybind11?

- Created by Wenzel Jakob, associate professor at EPFL
- Header-only C++11 library

What is pybind11?

- Created by Wenzel Jakob, associate professor at EPFL
- Header-only C++11 library
- Exposes C++ to Python and vice versa

What is pybind11?

- Created by Wenzel Jakob, associate professor at EPFL
- Header-only C++11 library
- Exposes C++ to Python and vice versa
- A convenient wrapper over Python's C API (**Python.h**)

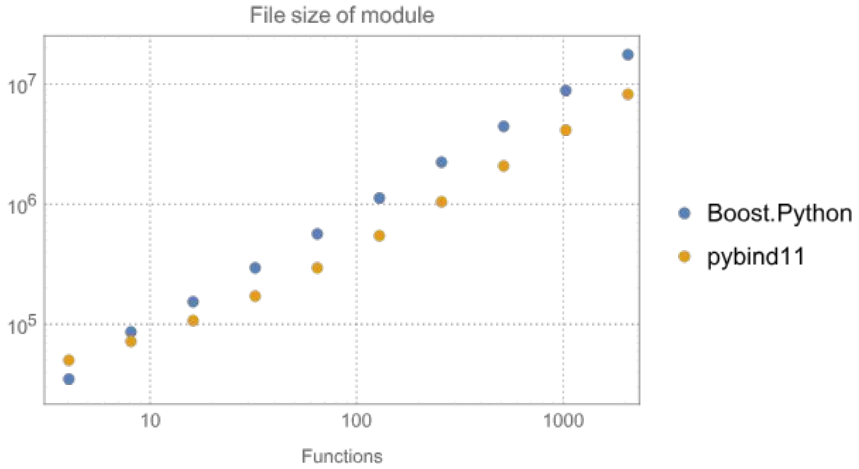
What is pybind11?

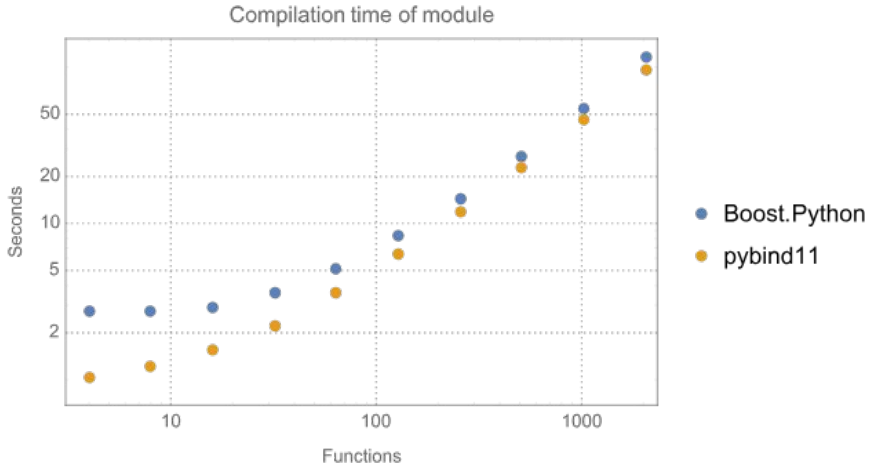
- Created by Wenzel Jakob, associate professor at EPFL
- Header-only C++11 library
- Exposes C++ to Python and vice versa
- A convenient wrapper over Python's C API (**Python.h**)
- Minimal boilerplate

What is pybind11?

pybind11

- Created by Wenzel Jakob, associate professor at EPFL
- Header-only C++11 library
- Exposes C++ to Python and vice versa
- A convenient wrapper over Python's C API (**Python.h**)
- Minimal boilerplate





Does anybody even use pybind11?

Does anybody even use pybind11?

Code comparison: Python.h vs pybind11

Code comparison: Python.h vs pybind11

```
3   void sort(int *a, size_t size) {
4       int curr;
5       int j;
6
7       for (size_t i = 1; i < size; i++) {
8           curr = a[i];
9           j = i - 1;
10
11          while (j >= 0 && a[j] > curr) {
12              a[j+1] = a[j];
13              j = j - 1;
14          }
15
16          a[j+1] = curr;
17      }
18  }
```

Code comparison: Python.h vs pybind11

```c
static PyObject *mod_sort(PyObject *self, PyObject *args) {
    PyObject *array = NULL;

    if (!PyArg_ParseTuple(args, "O", &array)) {
        return NULL;
    }

    int size = PyList_Size(array);
    int a[size];

    for (int i = 0; i < size; i++) {
        a[i] = PyLong_AsLong(PyList_GetItem(array, i));
    }

    PyObject *sorted = PyList_New(size);

    sort(a, size);

    for (int i = 0; i < size; i++) {
        PyList_SetItem(sorted, i, PyLong_FromLong(a[i]));
    }

    return Py_BuildValue("O", sorted);
}
```

```c
static PyMethodDef mod_methods[] = {
    {"sort", (PyCFunction)mod_sort, METH_VARARGS, "
    {NULL, NULL, 0, NULL}
};

static struct PyModuleDef modmodule = {
    PyModuleDef_HEAD_INIT,
    "mod",
    NULL,
    -1,
    mod_methods
};

PyMODINIT_FUNC PyInit_mod(void){
    return PyModule_Create(&modmodule);
}
```

Code comparison: Python.h vs pybind11

```
1   std::vector<int> sort_vec(std::vector<int> a) {
2       sort(a.data(), a.size());
3       return a;
4   }
5
6   PYBIND11_MODULE(sorting, m) {
7       m.def("sort", &sort_vec, "Insertion sort");
8   }
```

And what is nanobind?

And what is nanobind?

- Also created by Wenzel Jakob, as a successor to pybind11

And what is nanobind?

- Also created by Wenzel Jakob, as a successor to pybind11
- Fully utilizes C++17 features

And what is nanobind?

- Also created by Wenzel Jakob, as a successor to pybind11
- Fully utilizes C++17 features
- Faster compilation

And what is nanobind?

- Also created by Wenzel Jakob, as a successor to pybind11
- Fully utilizes C++17 features
- Faster compilation
- Smaller binary sizes

And what is nanobind?

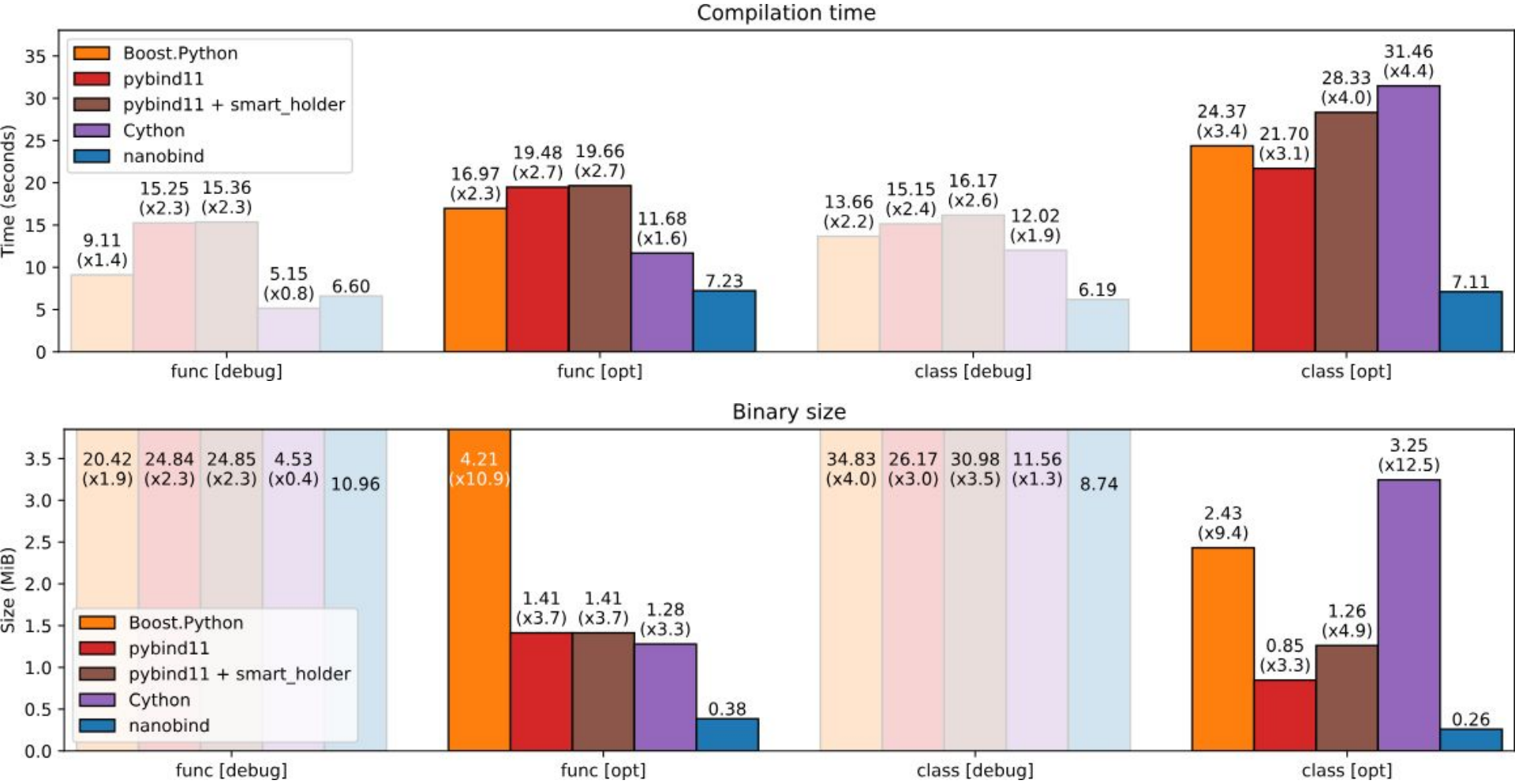
- Also created by Wenzel Jakob, as a successor to pybind11
- Fully utilizes C++17 features
- Faster compilation
- Smaller binary sizes
- Lower runtime overhead

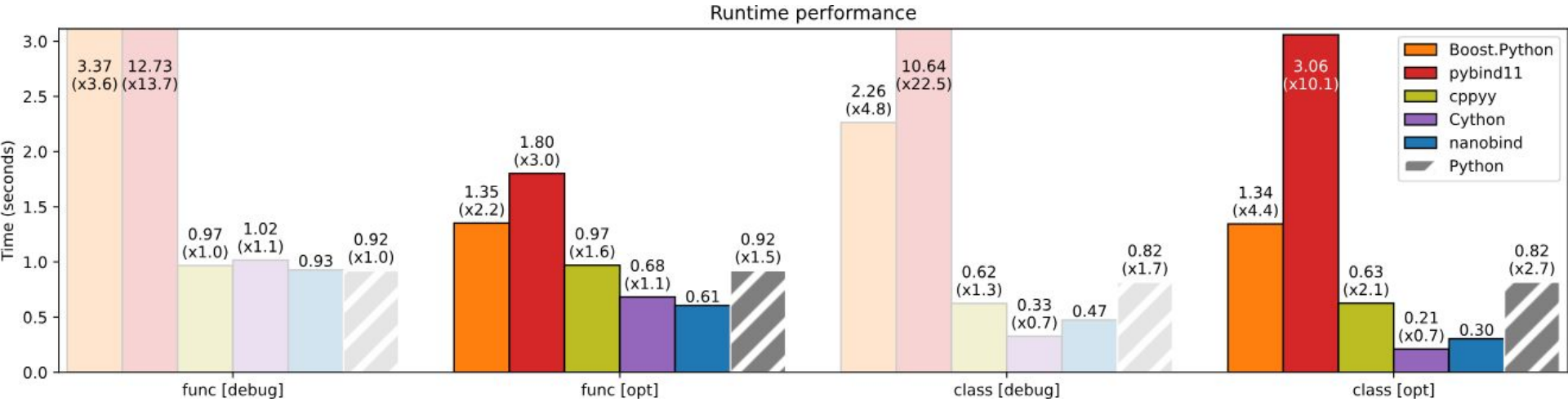
And what is nanobind?

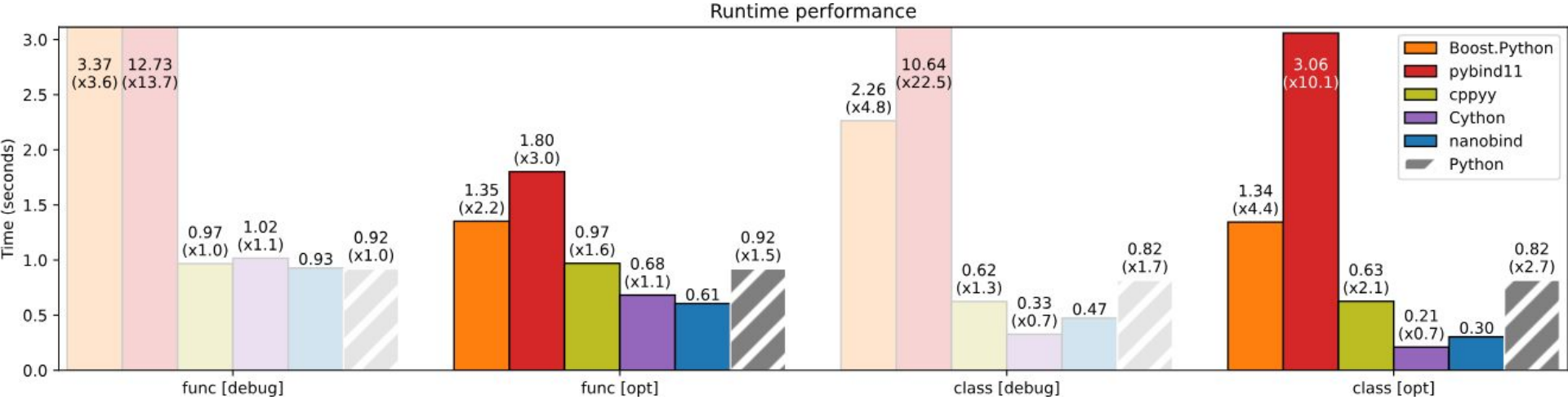
- Also created by Wenzel Jakob, as a successor to pybind11
- Fully utilizes C++17 features
- Faster compilation
- Smaller binary sizes
- Lower runtime overhead
- Very similar API

And what is nanobind?

- Also created by Wenzel Jakob, as a successor to pybind11
- Fully utilizes C++17 features
- Faster compilation
- Smaller binary sizes
- Lower runtime overhead
- Very similar API
- Build on top of years of feedback
  and lessons learned from pybind11


Compilation time
Time (seconds)
Boost.Python
pybind11
pybind11 + smart_holder
Cython
nanobind
9.11 (x1.4)
15.25 (x2.3)
15.36 (x2.3)
5.15 (x0.8)
6.60
16.97 (x2.3)
19.48 (x2.7)
19.66 (x2.7)
11.68 (x1.6)
7.23
13.66 (x2.2)
15.15 (x2.4)
16.17 (x2.6)
12.02 (x1.9)
6.19
24.37 (x3.4)
21.70 (x3.1)
28.33 (x4.0)
31.46 (x4.4)
7.11
func [debug]
func [opt]
class [debug]
class [opt]
Binary size
Size (MiB)
20.42 (x1.9)
24.84 (x2.3)
24.85 (x2.3)
4.53 (x0.4)
10.96
4.21 (x10.9)
1.41 (x3.7)
1.41 (x3.7)
1.28 (x3.3)
0.38
34.83 (x4.0)
26.17 (x3.0)
30.98 (x3.5)
11.56 (x1.3)
8.74
2.43 (x9.4)
0.85 (x3.3)
1.26 (x4.9)
3.25 (x12.5)
0.26
func [debug]
func [opt]
class [debug]
class [opt]


Runtime performance
Time (seconds)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
3.37 (x3.6)
12.73 (x13.7)
0.97 (x1.0)
1.02 (x1.1)
0.93
0.92 (x1.0)
1.35 (x2.2)
1.80 (x3.0)
0.97 (x1.6)
0.68 (x1.1)
0.61
0.92 (x1.5)
2.26 (x4.8)
10.64 (x22.5)
0.62 (x1.3)
0.33 (x0.7)
0.47
0.82 (x1.7)
1.34 (x4.4)
3.06 (x10.1)
0.63 (x2.1)
0.21 (x0.7)
0.30
0.82 (x2.7)
Boost.Python
pybind11
cppyy
Cython
nanobind
Python
func [debug]
func [opt]
class [debug]
class [opt]


Runtime performance
Time (seconds)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
3.37 (x3.6)
12.73 (x13.7)
0.97 (x1.0)
1.02 (x1.1)
0.93
0.92 (x1.0)
1.35 (x2.2)
1.80 (x3.0)
0.97 (x1.6)
0.68 (x1.1)
0.61
0.92 (x1.5)
2.26 (x4.8)
10.64 (x22.5)
0.62 (x1.3)
0.33 (x0.7)
0.47
0.82 (x1.7)
1.34 (x4.4)
3.06 (x10.1)
0.63 (x2.1)
0.21 (x0.7)
0.30
0.82 (x2.7)
Boost.Python
pybind11
cppyy
Cython
nanobind
Python
func [debug]
func [opt]
class [debug]
class [opt]

What is pybind11_json?

What is pybind11_json?

- Header-only library that bridges pybind11 and nlohmann/json

What is pybind11_json?

- Header-only library that bridges pybind11 and nlohmann/json
- Automatic conversion between **nlohmann::json** and Python types (dict, list, string, tuple etc.)

What is pybind11_json?

- Header-only library that bridges pybind11 and nlohmann/json
- Automatic conversion between **nlohmann::json** and Python types (dict, list, string, tuple etc.)
- Just include one header and it works! No manual conversion needed!

```cpp
#include <pybind11/pybind11.h>
#include <pybind11_json/pybind11_json.hpp>
#include <nlohmann/json.hpp>

nlohmann::json process(nlohmann::json data) {
    data["processed"] = true;
    return data;
}

PYBIND11_MODULE(example, m) {
    m.def("process", &process);
}
```

```cpp
#include <pybind11/pybind11.h>
#include <pybind11_json/pybind11_json.hpp>
#include <nlohmann/json.hpp>

nlohmann::json process(nlohmann::json data) {
    data["processed"] = true;
    return data;
}

PYBIND11_MODULE(example, m) {
    m.def("process", &process);
}
```

```
Python 3.12.3 (main, Mar 23 2026, 19:04:32) [GCC 13.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import example
>>> a = {"name": "Pykonik", "year": 2026}
>>> example.process(a)
{'name': 'Pykonik', 'processed': True, 'year': 2026}
>>>
```

Does anybody even use pybind11_json?

Does anybody even use pybind11_json?

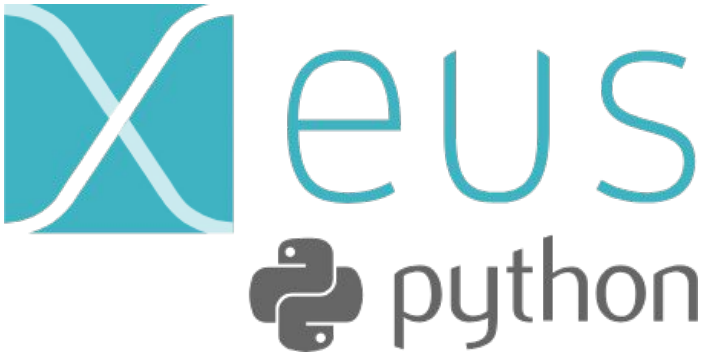
- Well…

Does anybody even use pybind11_json?

- Well… Yeah

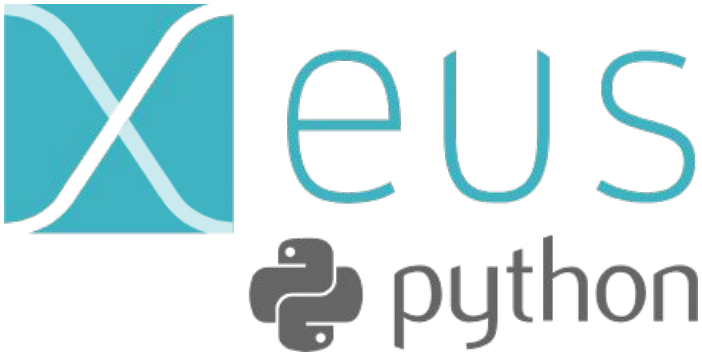
Does anybody even use pybind11_json?

- Well… Yeah



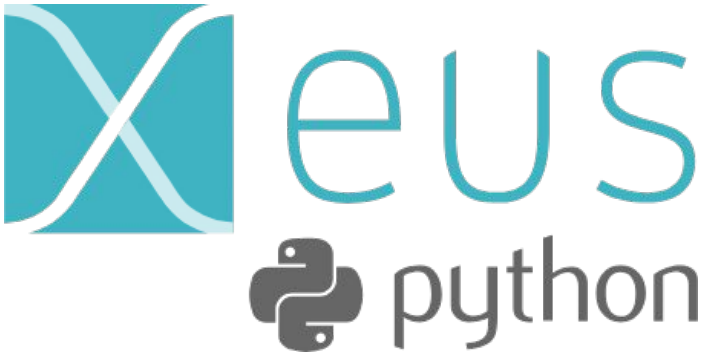
Does anybody even use pybind11_json?

- Well… Yeah



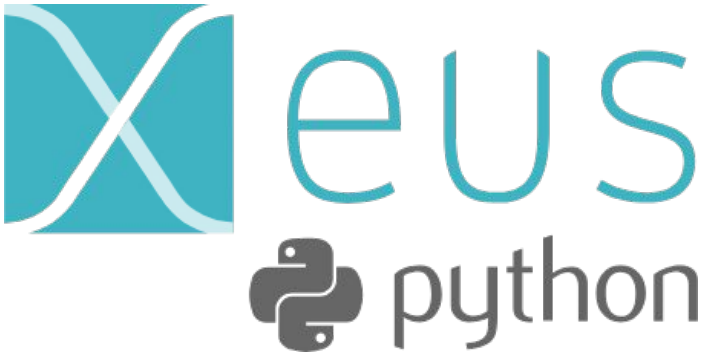
Does anybody even use pybind11_json?

- Well… Yeah



Does anybody even use pybind11_json?

- Well… Yeah



I contribute here!

Quickly on PyPartMC

Quickly on PyPartMC

- Pythonic wrapper for PartMC (“Particle-resolved Monte Carlo code for atmospheric aerosol simulation”)

Quickly on PyPartMC

- Pythonic wrapper for PartMC (“Particle-resolved Monte Carlo code for atmospheric aerosol simulation”)
- Developed at University of Illinois Urbana-Champaign and AGH University of Kraków

Quickly on PyPartMC

- Pythonic wrapper for PartMC (“Particle-resolved Monte Carlo code for atmospheric aerosol simulation”)
- Developed at University of Illinois Urbana-Champaign and AGH University of Kraków
- 4 languages in the codebase!

Quickly on PyPartMC

- Pythonic wrapper for PartMC (“Particle-resolved Monte Carlo code for atmospheric aerosol simulation”)
- Developed at University of Illinois Urbana-Champaign and AGH University of Kraków
- 4 languages in the codebase! (C, C++, Fortran, Python)

Quickly on PyPartMC

- Pythonic wrapper for PartMC (“Particle-resolved Monte Carlo code for atmospheric aerosol simulation”)
- Developed at University of Illinois Urbana-Champaign and AGH University of Kraków
- 4 languages in the codebase! (C, C++, Fortran, Python)
- Available on Linux, Windows and macOS

Quickly on PyPartMC

- Pythonic wrapper for PartMC (“Particle-resolved Monte Carlo code for atmospheric aerosol simulation”)
- Developed at University of Illinois Urbana-Champaign and AGH University of Kraków
- 4 languages in the codebase! (C, C++, Fortran, Python)
- Available on Linux, Windows and macOS
- Uses pybind11_json under the hood to provide a more Pythonic experience, and enable running without any auxiliary files

Quickly on PyPartMC

- Pythonic wrapper for PartMC (“Particle-resolved Monte Carlo code for atmospheric aerosol simulation”)
- Developed at University of Illinois Urbana-Champaign and AGH University of Kraków
- 4 languages in the codebase! (C, C++, Fortran, Python)
- Available on Linux, Windows and macOS
- Uses pybind11_json under the hood to provide a more Pythonic experience, and enable running without any auxiliary files
- We decided to switch to **nanobind**, we achieved an almost x4 speed-up on compilation!

So let’s switch!

So let’s switch!

- Changing pybind11 code to nanobind is relatively simple, most of the API is the same

So let’s switch!

- Changing pybind11 code to nanobind is relatively simple, most of the API is the same
- We need an alternative to pybind11_json

So let’s switch!

- Changing pybind11 code to nanobind is relatively simple, most of the API is the same
- We need an alternative to pybind11_json
- Luckily, it exists!

So let’s switch!

- Changing pybind11 code to nanobind is relatively simple, most of the API is the same
- We need an alternative to pybind11_json
- Luckily, it exists!

> Tip
>
> Looking for an equivalent with nanobind? Check this out https://github.com/ianhbell/nanobind_json !

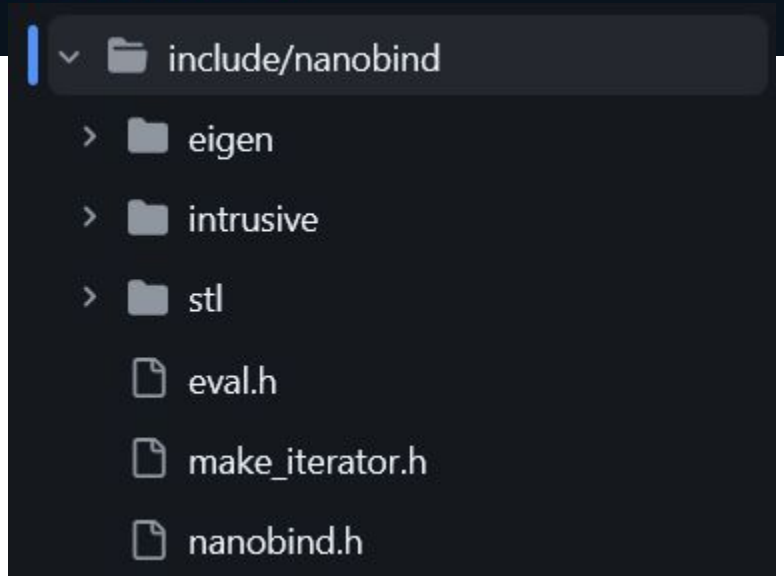
Hmm… just a small error

Hmm… just a small error

```
nanobind_json/include/nanobind_json/nanobind_json.hpp:16:10: fatal error: nanobind/nanob
ind.hpp: No such file or directory
   16 | #include "nanobind/nanobind.hpp"
      |          ^~~~~~~~~~~~~~~~~~~~~~~
compilation terminated.
```

Hmm… just a small error

```
nanobind_json/include/nanobind_json/nanobind_json.hpp:16:10: fatal error: nanobind/nanob
ind.hpp: No such file or directory
   16 | #include "nanobind/nanobind.hpp"
      |          ^~~~~~~~~~~~~~~~~~~~~~~
compilation terminated.
```



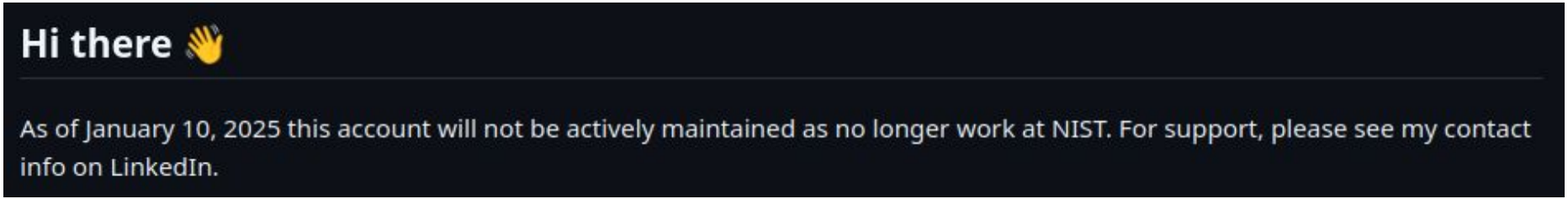
```
/home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nb_func.h: In instantiation of 'PyObject* nanobind::detail::func_create(Func&&, Return (*)(Args ...), std::index_sequence<Is
2 ...>, const Extra& ...) [with bool ReturnRef = false; bool CheckGuard = true; Func = void (*&)(nlohmann::json_abi_v3_12_0::basic_json<>); Return = void; Args = {nlohmann::json_abi_v3_12_0::b
asic_json<std::map, std::vector, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, bool, long int, long unsigned int, double, std::allocator, nlohmann::json_abi_
v3_12_0::adl_serializer, std::vector<unsigned char, std::allocator<unsigned char> >, void>}; long unsigned int ...Is = {0}; Extra = {nanobind::scope, nanobind::name}; PyObject = _object; std::
index_sequence<Is2 ...> = std::integer_sequence<long unsigned int, 0>]':
/home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nb_func.h:332:37:   required from 'void nanobind::cpp_function_def(Return (*)(Args ...), const Extra& ...) [with <template-p
arameter-1-1> = void; Return = void; Args = {nlohmann::json_abi_v3_12_0::basic_json<std::map, std::vector, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, bool
, long int, long unsigned int, double, std::allocator, nlohmann::json_abi_v3_12_0::adl_serializer, std::vector<unsigned char, std::allocator<unsigned char> >, void>}; Extra = {scope, name}]'
/home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nb_func.h:409:21:   required from 'nanobind::module_& nanobind::module_::def(const char*, Func&&, const Extra& ...) [with Fu
nc = void (*)(nlohmann::json_abi_v3_12_0::basic_json<>); Extra = {}]'
/home/graca/Desktop/temp/temp1/bindings.cpp:25:10:   required from here
/home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nb_func.h:178:64: error: 'Name' is not a member of 'nanobind::detail::make_caster<nlohmann::json_abi_v3_12_0::basic_json<> >
' {aka 'nanobind::detail::type_caster<nlohmann::json_abi_v3_12_0::basic_json<> >'}
  178 |             make_caster<remove_opt_mono_t<intrinsic_t<Args>>>::Name)...) +
      |                                                               ^~~~
/home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nb_func.h:254:41: error: 'struct nanobind::detail::type_caster<nlohmann::json_abi_v3_12_0::basic_json<> >' has no member nam
ed 'from_python'
  254 |             if ((!in.template get<Is>().from_python(args[Is], args_flags[Is],
      |                    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~~~~~~~~~~~
In file included from /home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nanobind.h:53:
/home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nb_cast.h: In substitution of 'template<class T> using nanobind::detail::cast_t = typename nanobind::detail::make_caster<T>:
:Cast<T> [with T = nlohmann::json_abi_v3_12_0::basic_json<>]':
/home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nb_func.h:264:54:   required from 'PyObject* nanobind::detail::func_create(Func&&, Return (*)(Args ...), std::index_sequence
<Is2 ...>, const Extra& ...) [with bool ReturnRef = false; bool CheckGuard = true; Func = void (*&)(nlohmann::json_abi_v3_12_0::basic_json<>); Return = void; Args = {nlohmann::json_abi_v3_12_0
::basic_json<std::map, std::vector, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, bool, long int, long unsigned int, double, std::allocator, nlohmann::json_a
bi_v3_12_0::adl_serializer, std::vector<unsigned char, std::allocator<unsigned char> >, void>}; long unsigned int ...Is = {0}; Extra = {nanobind::scope, nanobind::name}; PyObject = _object; st
d::index_sequence<Is2 ...> = std::integer_sequence<long unsigned int, 0>]'
/home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nb_func.h:332:37:   required from 'void nanobind::cpp_function_def(Return (*)(Args ...), const Extra& ...) [with <template-p
arameter-1-1> = void; Return = void; Args = {nlohmann::json_abi_v3_12_0::basic_json<std::map, std::vector, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, bool
, long int, long unsigned int, double, std::allocator, nlohmann::json_abi_v3_12_0::adl_serializer, std::vector<unsigned char, std::allocator<unsigned char> >, void>}; Extra = {scope, name}]'
/home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nb_func.h:409:21:   required from 'nanobind::module_& nanobind::module_::def(const char*, Func&&, const Extra& ...) [with Fu
nc = void (*)(nlohmann::json_abi_v3_12_0::basic_json<>); Extra = {}]'
/home/graca/Desktop/temp/temp1/bindings.cpp:25:10:   required from here
/home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nb_cast.h:59:29: error: no class template named 'Cast' in 'nanobind::detail::make_caster<nlohmann::json_abi_v3_12_0::basic_j
son<> >' {aka 'struct nanobind::detail::type_caster<nlohmann::json_abi_v3_12_0::basic_json<> >'}
   59 | template <typename T> using cast_t = typename make_caster<T>::template Cast<T>;
      |                             ^~~~~~
```

The problem begins… 439 lines of errors!

```
/home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nb_func.h: In instantiation of 'PyObject* nanobind::detail::func_create(Func&&, Return (*)(Args ...), std::index_sequence<Is
2 ...>, const Extra& ...) [with bool ReturnRef = false; bool CheckGuard = true; Func = void (*&)(nlohmann::json_abi_v3_12_0::basic_json<>); Return = void; Args = {nlohmann::json_abi_v3_12_0::b
asic_json<std::map, std::vector, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, bool, long int, long unsigned int, double, std::allocator, nlohmann::json_abi_
v3_12_0::adl_serializer, std::vector<unsigned char, std::allocator<unsigned char> >, void>}; long unsigned int ...Is = {0}; Extra = {nanobind::scope, nanobind::name}; PyObject = _object; std::
index_sequence<Is2 ...> = std::integer_sequence<long unsigned int, 0>]':
/home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nb_func.h:332:37:   required from 'void nanobind::cpp_function_def(Return (*)(Args ...), const Extra& ...) [with <template-p
arameter-1-1> = void; Return = void; Args = {nlohmann::json_abi_v3_12_0::basic_json<std::map, std::vector, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, bool
, long int, long unsigned int, double, std::allocator, nlohmann::json_abi_v3_12_0::adl_serializer, std::vector<unsigned char, std::allocator<unsigned char> >, void>}; Extra = {scope, name}]'
/home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nb_func.h:409:21:   required from 'nanobind::module_& nanobind::module_::def(const char*, Func&&, const Extra& ...) [with Fu
nc = void (*)(nlohmann::json_abi_v3_12_0::basic_json<>); Extra = {}]'
/home/graca/Desktop/temp/temp1/bindings.cpp:25:10:   required from here
/home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nb_func.h:178:64: error: 'Name' is not a member of 'nanobind::detail::make_caster<nlohmann::json_abi_v3_12_0::basic_json<> >
' {aka 'nanobind::detail::type_caster<nlohmann::json_abi_v3_12_0::basic_json<> >'}
  178 |             make_caster<remove_opt_mono_t<intrinsic_t<Args>>>::Name)...) +
      |                                                                ^~~~
/home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nb_func.h:254:41: error: 'struct nanobind::detail::type_caster<nlohmann::json_abi_v3_12_0::basic_json<> >' has no member nam
ed 'from_python'
  254 |             if ((!in.template get<Is>().from_python(args[Is], args_flags[Is],
      |                   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~~~~~~~~~~~
In file included from /home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nanobind.h:53:
/home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nb_cast.h: In substitution of 'template<class T> using nanobind::detail::cast_t = typename nanobind::detail::make_caster<T>:
:Cast<T> [with T = nlohmann::json_abi_v3_12_0::basic_json<>]':
/home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nb_func.h:264:54:   required from 'PyObject* nanobind::detail::func_create(Func&&, Return (*)(Args ...), std::index_sequence
<Is2 ...>, const Extra& ...) [with bool ReturnRef = false; bool CheckGuard = true; Func = void (*&)(nlohmann::json_abi_v3_12_0::basic_json<>); Return = void; Args = {nlohmann::json_abi_v3_12_0
::basic_json<std::map, std::vector, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, bool, long int, long unsigned int, double, std::allocator, nlohmann::json_a
bi_v3_12_0::adl_serializer, std::vector<unsigned char, std::allocator<unsigned char> >, void>}; long unsigned int ...Is = {0}; Extra = {nanobind::scope, nanobind::name}; PyObject = _object; st
d::index_sequence<Is2 ...> = std::integer_sequence<long unsigned int, 0>]'
/home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nb_func.h:332:37:   required from 'void nanobind::cpp_function_def(Return (*)(Args ...), const Extra& ...) [with <template-p
arameter-1-1> = void; Return = void; Args = {nlohmann::json_abi_v3_12_0::basic_json<std::map, std::vector, std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >, bool
, long int, long unsigned int, double, std::allocator, nlohmann::json_abi_v3_12_0::adl_serializer, std::vector<unsigned char, std::allocator<unsigned char> >, void>}; Extra = {scope, name}]'
/home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nb_func.h:409:21:   required from 'nanobind::module_& nanobind::module_::def(const char*, Func&&, const Extra& ...) [with Fu
nc = void (*)(nlohmann::json_abi_v3_12_0::basic_json<>); Extra = {}]'
/home/graca/Desktop/temp/temp1/bindings.cpp:25:10:   required from here
/home/graca/Desktop/temp/temp1/gitmodules/nanobind/include/nanobind/nb_cast.h:59:29: error: no class template named 'Cast' in 'nanobind::detail::make_caster<nlohmann::json_abi_v3_12_0::basic_j
son<> >' {aka 'struct nanobind::detail::type_caster<nlohmann::json_abi_v3_12_0::basic_json<> >'}
   59 | template <typename T> using cast_t = typename make_caster<T>::template Cast<T>;
      |                                                   ^~~~~
```

Let’s get in touch with the maintainer!

Let’s get in touch with the maintainer!

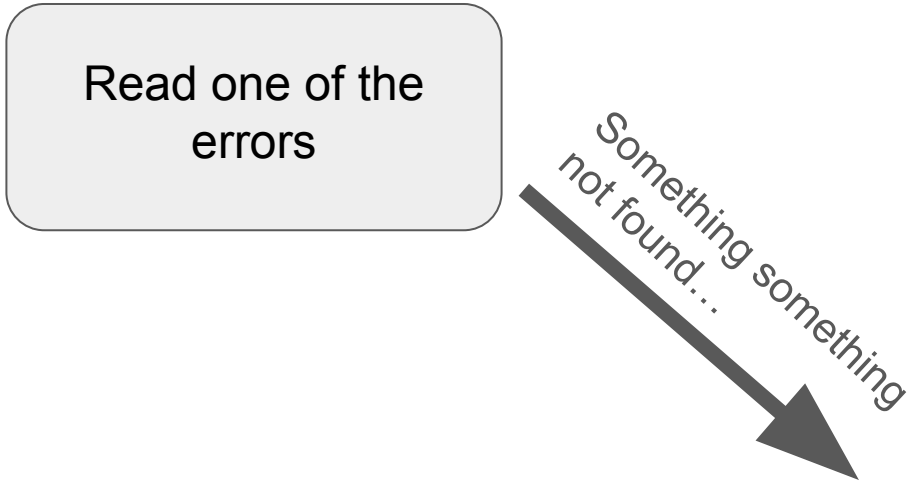


So let’s fix it ourselves!

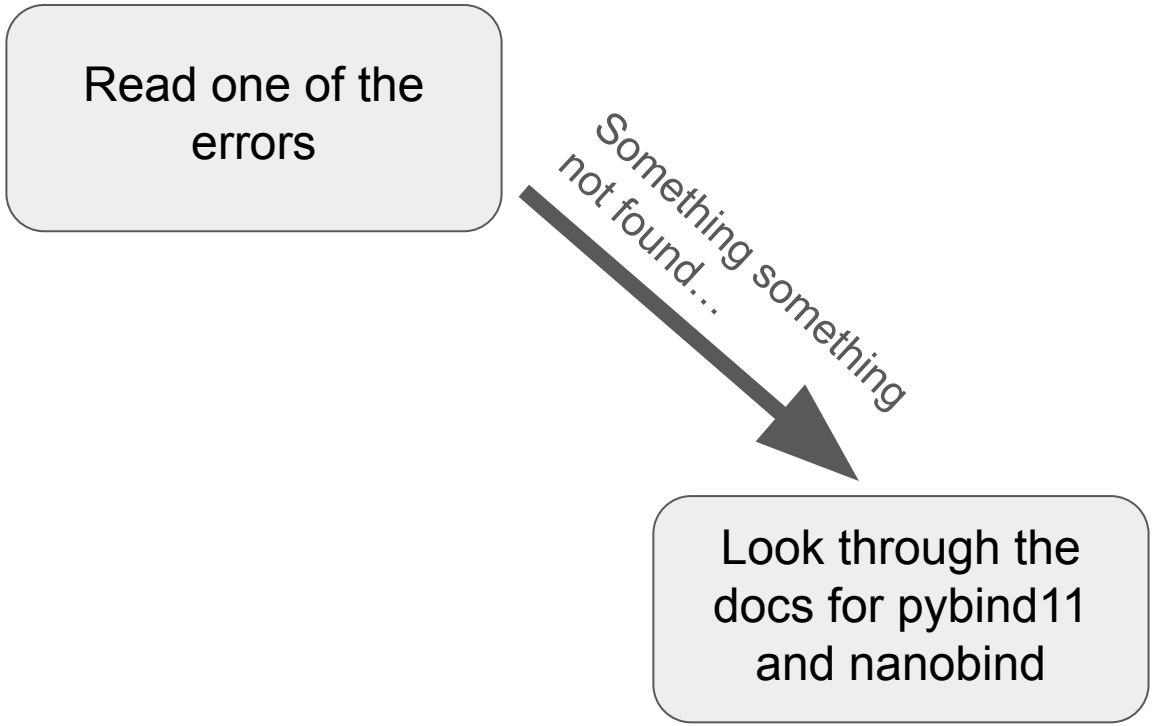
So let’s fix it ourselves!

Read one of the errors

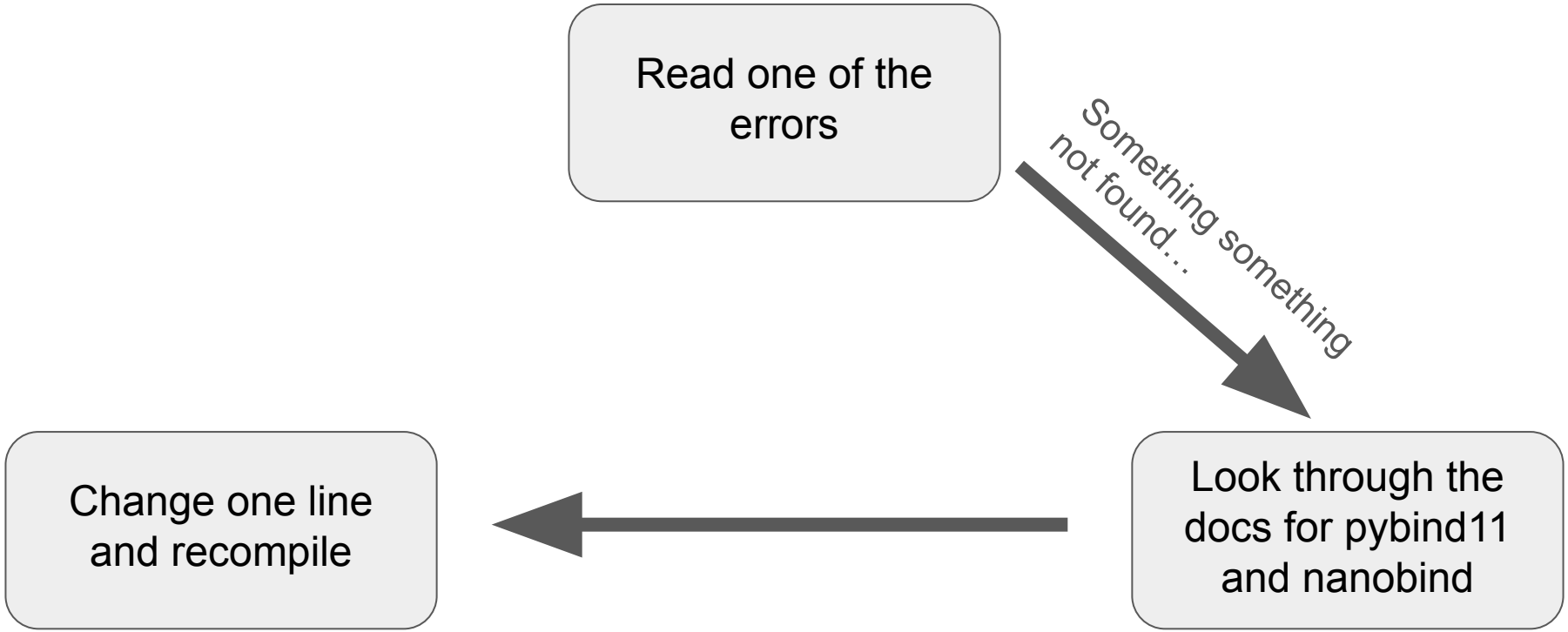
So let’s fix it ourselves!



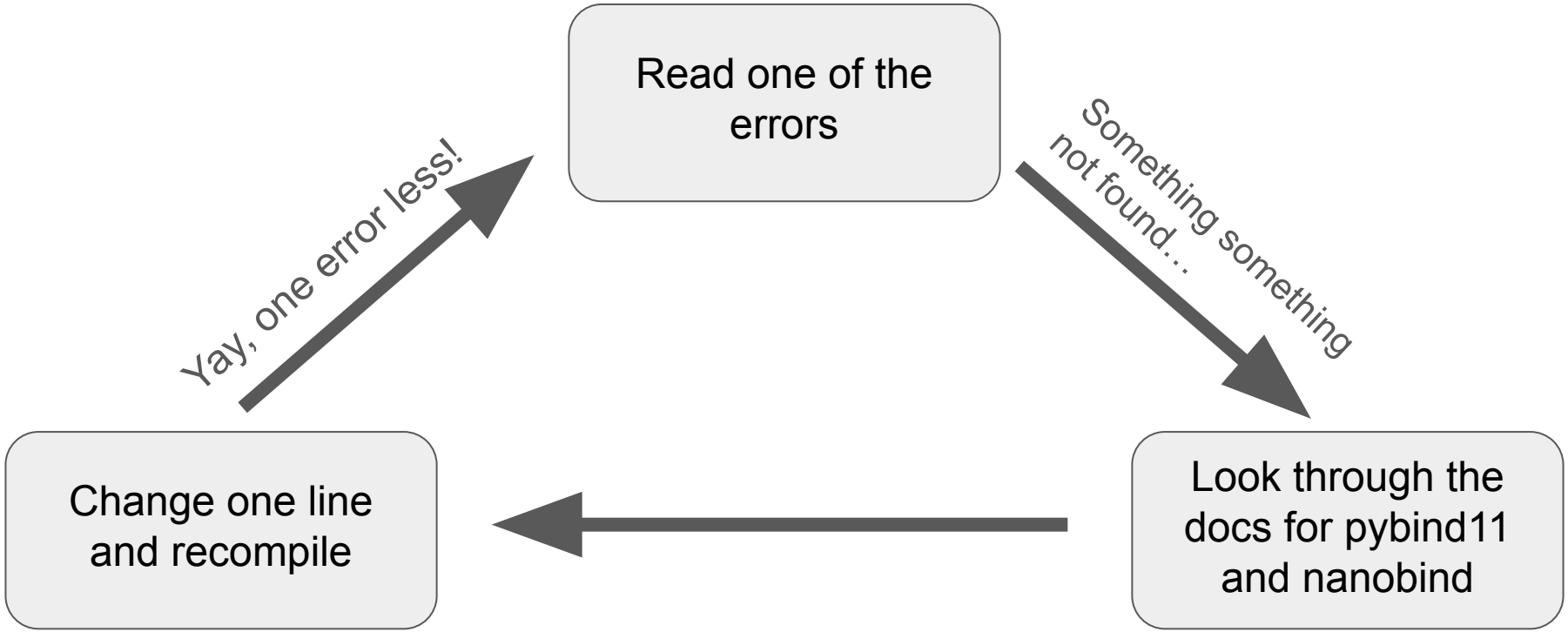
So let’s fix it ourselves!



So let’s fix it ourselves!



So let’s fix it ourselves!



Example: cast()

Example: cast()

```
class handle : public detail::object_api<handle>
```

Holds a reference to a Python object (no reference counting)

```
template<typename T>
    T cast() const
```

Attempt to cast the Python object into the given C++ type. A `cast_error` will be throw upon failure.

Example: cast()

```
class handle : public detail::object_api<handle>
```

Holds a reference to a Python object (no reference counting)

```
template<typename T>
    T cast() const
```

Attempt to cast the Python object into the given C++ type. A `cast_error` will be throw upon failure.

```
pybind11::handle a = pybind11::int_(5);
int b = a.cast<int>();
```

Example: cast()

```
template<typename T, typename Derived>
    T cast(const detail::api<Derived> &value, bool convert = true)
```

Convert the Python object `value` (typically a `handle` or a `object` subclass) into a C++ object of type `T`.

```
[](nb::object a) {
    int b = nb::cast<int>(a);
    return b + 1;
};
```

Example: load() and cast()

Example: load() and cast()

Python to C++

```
bool load(handle src,
    bool convert) {
```

C++ to Python

```
static handle cast(
    const YourType &number,
    return_value_policy policy,
    handle parent) {
```

Example: load() and cast()

Python to C++

```
bool load(handle src,
    bool convert) {
```

Python to C++

```
bool from_python(handle src,
    uint8_t flags,
    cleanup_list *cleanup)
    noexcept {
```

C++ to Python

```
static handle cast(
    const YourType &number,
    return_value_policy policy,
    handle parent) {
```

C++ to Python

```
static handle from_cpp(
    YourType &&value,
    rv_policy policy,
    cleanup_list *cleanup)
    noexcept {
```

Other problems:

Other problems: Circular References

Other problems: Circular References

```python
@staticmethod
@pytest.mark.skipif(platform.machine() == "arm64", reason="TODO #348")
def test_fixed_segfault_case_on_circular_reference():
    # arrange
    aero_data = ppmc.AeroData(AERO_DATA_CTOR_ARG_MINIMAL)
    fishy_ctor_arg = copy.deepcopy(AERO_MODE_CTOR_LOG_NORMAL)
    fishy_ctor_arg["test_mode"]["mass_frac"].append(
        fishy_ctor_arg["test_mode"]["mass_frac"]
    )

    # act
    with pytest.raises(RuntimeError) as exc_info:
        ppmc.AeroMode(aero_data, fishy_ctor_arg)
```

Other problems: Circular References

```python
@staticmethod
@pytest.mark.skipif(platform.machine() == "arm64", reason="TODO #348")
def test_fixed_segfault_case_on_circular_reference():
    # arrange
    aero_data = ppmc.AeroData(AERO_DATA_CTOR_ARG_MINIMAL)
    fishy_ctor_arg = copy.deepcopy(AERO_MODE_CTOR_LOG_NORMAL)
    fishy_ctor_arg["test_mode"]["mass_frac"].append(
        fishy_ctor_arg["test_mode"]["mass_frac"]
    )
```

Avoid crash when converting dict with circular reference #74

Merged martinRenou merged 1 commit into pybind:master from dbaston:fix-73 on Dec 9, 2024

Other problems: Circular References

```
inline nl::json to_json(const py::handle& obj, std::set<const PyObject*>& refs)
{
```

Other problems: Circular References

```
inline nl::json to_json(const py::handle& obj, std::set<const PyObject*>& refs)
{
    auto insert_ret = refs.insert(obj.ptr());
    if (!insert_ret.second) {
        throw std::runtime_error("Circular reference detected");
    }
```

Other problems: Circular References

```
inline nl::json to_json(const py::handle& obj, std::set<const PyObject*>& refs)
{
    auto insert_ret = refs.insert(obj.ptr());
    if (!insert_ret.second) {
        throw std::runtime_error("Circular reference detected");
    }

    refs.erase(insert_ret.first);
```

Everything works!

Everything works!

- PyPartMC compiles

Everything works!

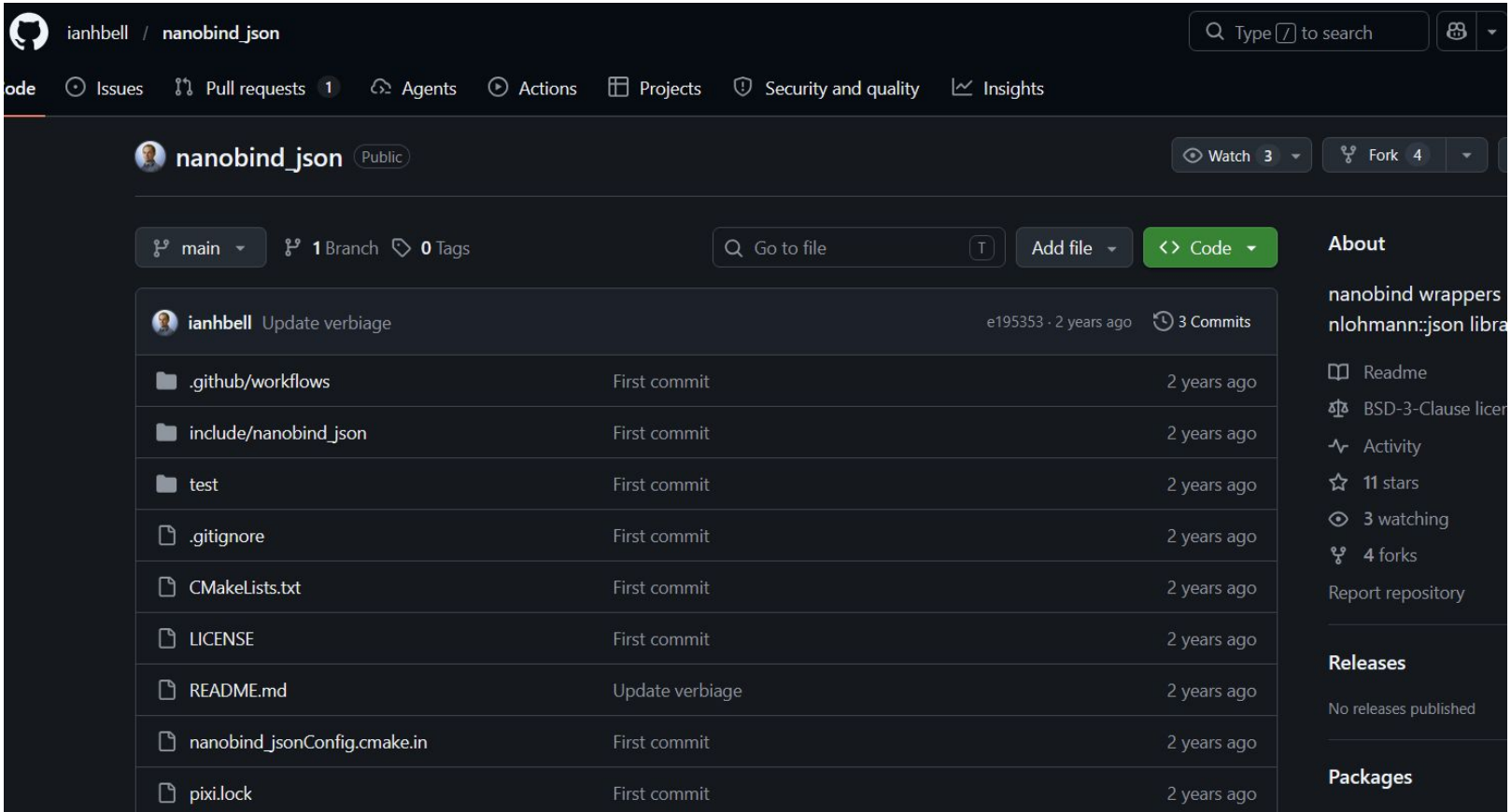
- PyPartMC compiles
- All 400 tests are passing

Everything works!

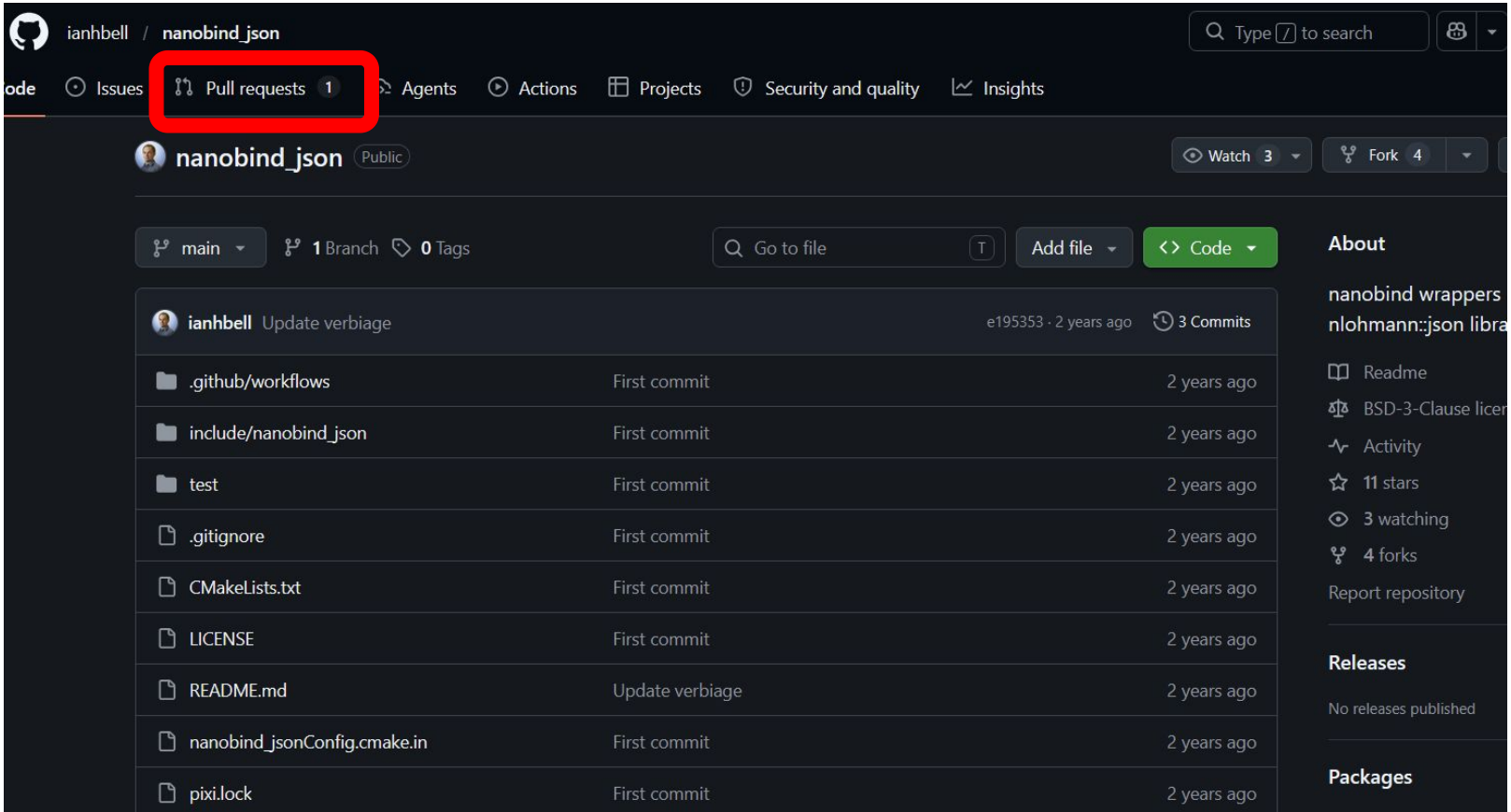
- PyPartMC compiles
- All 400 tests are passing
- Let’s fork nanobind_json and add our changes!

Time to fork

Time to fork


ianhbell / nanobind_json
Type / to search
ode
Issues
Pull requests 1
Agents
Actions
Projects
Security and quality
Insights
nanobind_json
Public
Watch 3
Fork 4
main
1 Branch
0 Tags
Go to file
T
Add file
Code
About
nanobind wrappers
nlohmann::json libra
ianhbell Update verbiage
e195353 · 2 years ago
3 Commits
.github/workflows
First commit
2 years ago
include/nanobind_json
First commit
2 years ago
test
First commit
2 years ago
.gitignore
First commit
2 years ago
CMakeLists.txt
First commit
2 years ago
LICENSE
First commit
2 years ago
README.md
Update verbiage
2 years ago
nanobind_jsonConfig.cmake.in
First commit
2 years ago
pixi.lock
First commit
2 years ago
Readme
BSD-3-Clause licer
Activity
11 stars
3 watching
4 forks
Report repository
Releases
No releases published
Packages

Time to fork


ianhbell / nanobind_json
Type / to search
ode
Issues
Pull requests 1
Agents
Actions
Projects
Security and quality
Insights
nanobind_json
Public
Watch 3
Fork 4
main
1 Branch
0 Tags
Go to file T
Add file
Code
About
nanobind wrappers
nlohmann::json libra
ianhbell Update verbiage
e195353 · 2 years ago
3 Commits
.github/workflows First commit 2 years ago
include/nanobind_json First commit 2 years ago
test First commit 2 years ago
.gitignore First commit 2 years ago
CMakeLists.txt First commit 2 years ago
LICENSE First commit 2 years ago
README.md Update verbiage 2 years ago
nanobind_jsonConfig.cmake.in First commit 2 years ago
pixi.lock First commit 2 years ago
Readme
BSD-3-Clause licer
Activity
11 stars
3 watching
4 forks
Report repository
Releases
No releases published
Packages

Time to fork

Time to fork


Update for nanobind v2.5.0
#1 opened on Mar 25, 2025 by pearzt


pearzt commented on Mar 25, 2025
I'm not particularly familiar with the internals of neither nlohmann_json nor nanobind, but I managed to update this for nanobind v2.5.0 and make it work for my use case. I'm posting this here just in case it's helpful for other people.
1

The Plan:

The Plan:

- Create a fork, add a temporary branch with the working setup for PyPartMC

The Plan:

- Create a fork, add a temporary branch with the working setup for PyPartMC
- Merge pearzt’s PR

The Plan:

- Create a fork, add a temporary branch with the working setup for PyPartMC
- Merge pearzt's PR
- Adjust where necessary

The Plan:

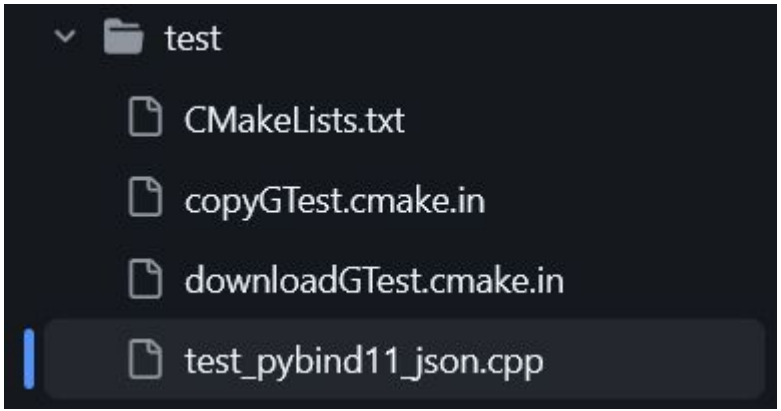
- Create a fork, add a temporary branch with the working setup for PyPartMC
- Merge pearzt's PR
- Adjust where necessary
- Write tests, covering the same scope as pybind11_json

The Plan:

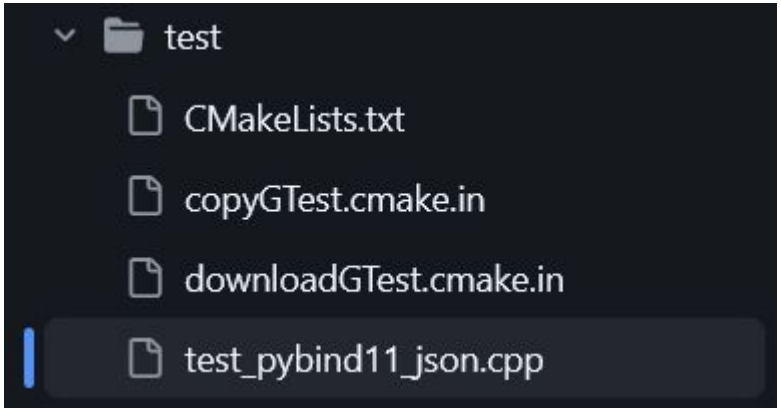
- Create a fork, add a temporary branch with the working setup for PyPartMC
- Merge pearzt's PR
- Adjust where necessary
- Write tests, covering the same scope as pybind11_json
- Add CI workflow for testing

Problem: No embedded interpreter

Problem: No embedded interpreter

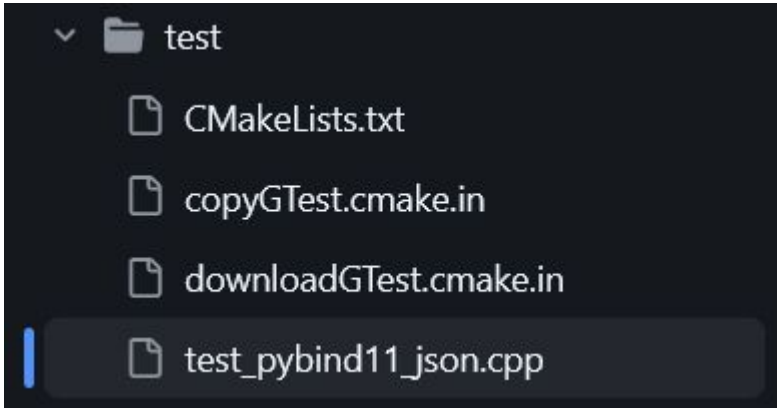


Problem: No embedded interpreter




```
#include "pybind11/embed.h"
```

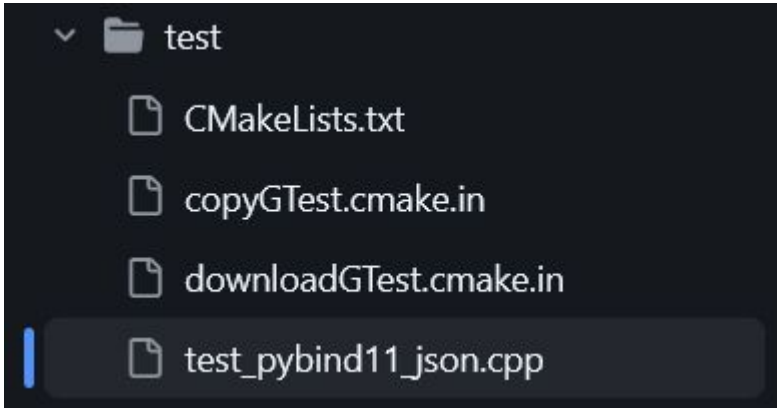
Problem: No embedded interpreter




```cpp
#include "pybind11/embed.h"
```

```cpp
TEST(nljson_serializers_tojson, bool_)
{
    py::scoped_interpreter guard;
    py::bool_ obj(false);
    nl::json j = obj;

    ASSERT_TRUE(j.is_boolean());
    ASSERT_FALSE(j.get<bool>());
}
```

Problem: No embedded interpreter

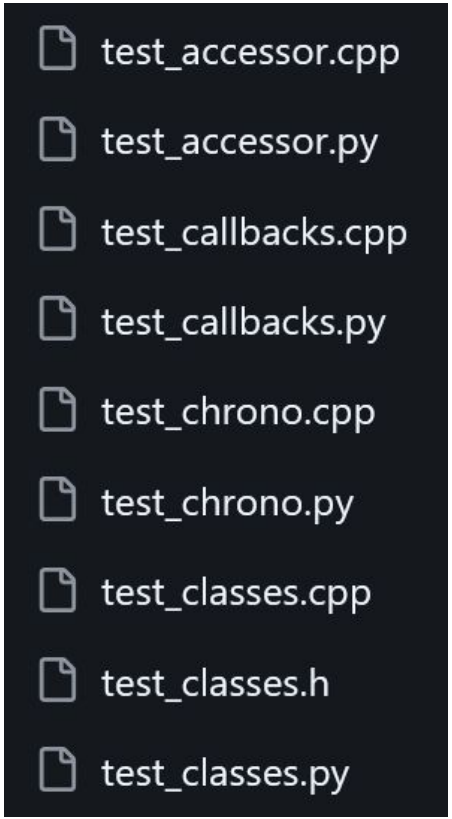



```
#include "pybind11/embed.h"
```

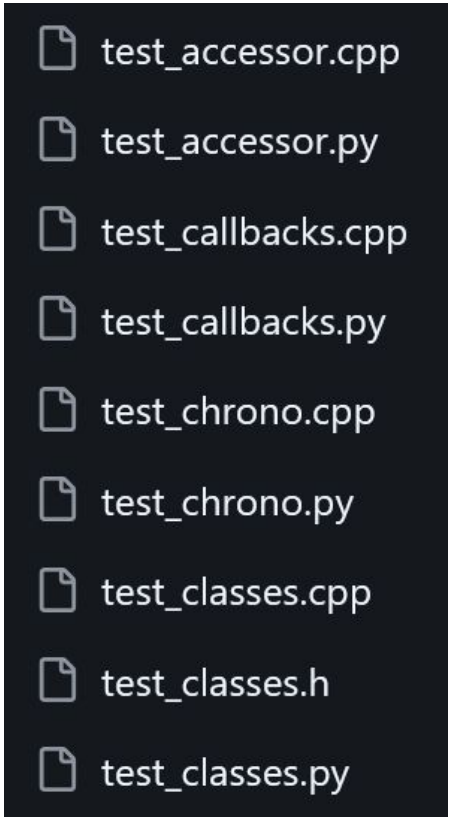
```
TEST(nljson_serializers_tojson, bool_)
{
    py::scoped_interpreter guard;
    py::bool_ obj(false);
    nl::json j = obj;

    ASSERT_TRUE(j.is_boolean());
    ASSERT_FALSE(j.get<bool>());
}
```

- ○ **Multi-intepreter, Embedding**: The ability to embed Python in an executable or run several independent Python interpreters in the same process is unsupported. Nanobind caters to bindings only. Multi-interpreter support would require TLS lookups for nanobind data structures, which is undesirable.

Problem: No embedded interpreter



Problem: No embedded interpreter




```
m.def("vec_copyable_in_value", [](std::vector<Copyable> x) {
    if (x.size() != 10)
        fail();
    for (int i = 0; i< 10; ++i)
        if (x[i].value != i)
            fail();
});
```

Problem: No embedded interpreter

- test_accessor.cpp
- test_accessor.py
- test_callbacks.cpp
- test_callbacks.py
- test_chrono.cpp
- test_chrono.py
- test_classes.cpp
- test_classes.h
- test_classes.py

```
m.def("vec_copyable_in_value", [](std::vector<Copyable> x) {
    if (x.size() != 10)
        fail();
    for (int i = 0; i< 10; ++i)
        if (x[i].value != i)
            fail();
});
```

```
def test25_vec_movable_in_value(clean):
    t.vec_copyable_in_value([t.Copyable(i) for i in range(10)])
    assert_stats(value_constructed=10, copy_constructed=10, destructed=20)
```

First test: Simple boolean

First test: Simple boolean

```
m.def("nljson_bool_fromjson", []() {
    return "false"_json;
});

m.def("nljson_bool_tojson", [](nl::json bool_json) {
    if (!bool_json.is_boolean()) {
        fail();
    }
    else if (bool_json.get<bool>() != false) {
        fail();
    }
});
```

First test: Simple boolean

```
m.def("nljson_bool_fromjson", []() {
    return "false"_json;
});

m.def("nljson_bool_tojson", [](nl::json bool_json) {
    if (!bool_json.is_boolean()) {
        fail();
    }
    else if (bool_json.get<bool>() != false) {
        fail();
    }
});
```

```
def test_nljson_bool_from_json():
    json = t.nljson_bool_fromjson()

    assert isinstance(json, bool)
    assert json == False

def test_nljson_bool_to_json():
    t.nljson_bool_tojson(False)
```

First test: Simple boolean

```
rootdir: /home/gadamus/zzz/nanobind_json
collected 2 items

../../tests/test_json.py::test_nljson_bool_from_json PASSED [ 50%]
../../tests/test_json.py::test_nljson_bool_to_json FAILED  [100%]

============================ FAILURES ============================
____________________ test_nljson_bool_to_json ____________________

    def test_nljson_bool_to_json():
>       t.nljson_bool_tojson(False)
E       RuntimeError: std::exception

../../tests/test_json.py:22: RuntimeError
==================== short test summary info =====================
FAILED ../../tests/test_json.py::test_nljson_bool_to_json - Runtim
eError: std::exception
================== 1 failed, 1 passed in 0.03s ===================
```

First test: Simple boolean

```
m.def("nljson_bool_tojson", [](nl::json bool_json) {
    if (!bool_json.is_boolean()) {
        std::cerr << "is not boolean\n";
        fail();
    }
    else if (bool_json.get<bool>() != false) {
        std::cerr << "is not false\n";
        fail();
    }
});
```

First test: Simple boolean

```
rootdir: /home/gadamus/zzz/nanobind_json
collected 2 items

../../tests/test_json.py::test_nljson_bool_from_json PASSED [ 50%]
../../tests/test_json.py::test_nljson_bool_to_json FAILED  [100%]

============================ FAILURES ============================
____________________ test_nljson_bool_to_json ____________________

    def test_nljson_bool_to_json():
>       t.nljson_bool_tojson(False)
E       RuntimeError: std::exception

../../tests/test_json.py:22: RuntimeError
---------------------- Captured stderr call ----------------------
is not boolean
==================== short test summary info =====================
FAILED ../../tests/test_json.py::test_nljson_bool_to_json - Runtim
eError: std::exception
================== 1 failed, 1 passed in 0.03s ===================
```

First test: Simple boolean

```
m.def("nljson_bool_tojson", [](nl::json bool_json) {
    std::cerr << bool_json.dump() << '\n';
    if (!bool_json.is_boolean()) {
        fail();
    }
    else if (bool_json.get<bool>() != false) {
        fail();
    }
});
```

First test: Simple boolean

```
rootdir: /home/gadamus/zzz/nanobind_json
collected 2 items

../../tests/test_json.py::test_nljson_bool_from_json PASSED [ 50%]
../../tests/test_json.py::test_nljson_bool_to_json FAILED  [100%]

============================ FAILURES ============================
____________________ test_nljson_bool_to_json ____________________

    def test_nljson_bool_to_json():
>       t.nljson_bool_tojson(False)
E       RuntimeError: std::exception

../../tests/test_json.py:22: RuntimeError
---------------------- Captured stderr call ----------------------
null
==================== short test summary info =====================
FAILED ../../tests/test_json.py::test_nljson_bool_to_json - Runtim
eError: std::exception
================== 1 failed, 1 passed in 0.03s ===================
```

What’s going on?

What’s going on?

- Only **from_python()** fails

What’s going on?

- Only **from_python()** fails
- PyPartMC works fine, despite sending JSONs from Python to C++

What’s going on?

- Only **from_python()** fails
- PyPartMC works fine, despite sending JSONs from Python to C++
- Receives “null” instead of bool

What’s going on?

- Only **from_python()** fails
- PyPartMC works fine, despite sending JSONs from Python to C++
- Receives “null” instead of bool
- Sending dictionaries or lists also fails

What’s going on?

- Only **from_python()** fails
- PyPartMC works fine, despite sending JSONs from Python to C++
- Receives “null” instead of bool
- Sending dictionaries or lists also fails
- So what’s wrong?

What’s going on?

- Only **from_python()** fails
- PyPartMC works fine, despite sending JSONs from Python to C++
- Receives “null” instead of bool
- Sending dictionaries or lists also fails
- So what’s wrong?
- Oh, PyPartMC uses a different branch, where we didn’t merge that other pull request

The culprit revealed

The culprit revealed

```
bool from_python(handle src, uint8_t flags, cleanup_list *cleanup) {
    try {
        auto value = nbjson::to_json<nl::json>(src);
        return true;
    }
    catch (nbjson::CircularReferenceError &e) {
        throw e;
    }
    catch (...) {
        return false;
    }
}
```

The culprit revealed

```
bool from_python(handle src, uint8_t flags, cleanup_list *cleanup) {
    try {
        auto value = nbjson::to_json<nl::json>(src);
        return true;
    }
    catch (nbjson::CircularReferenceError &e) {
        throw e;
    }
    catch (...) {
        return false;
    }
}
```

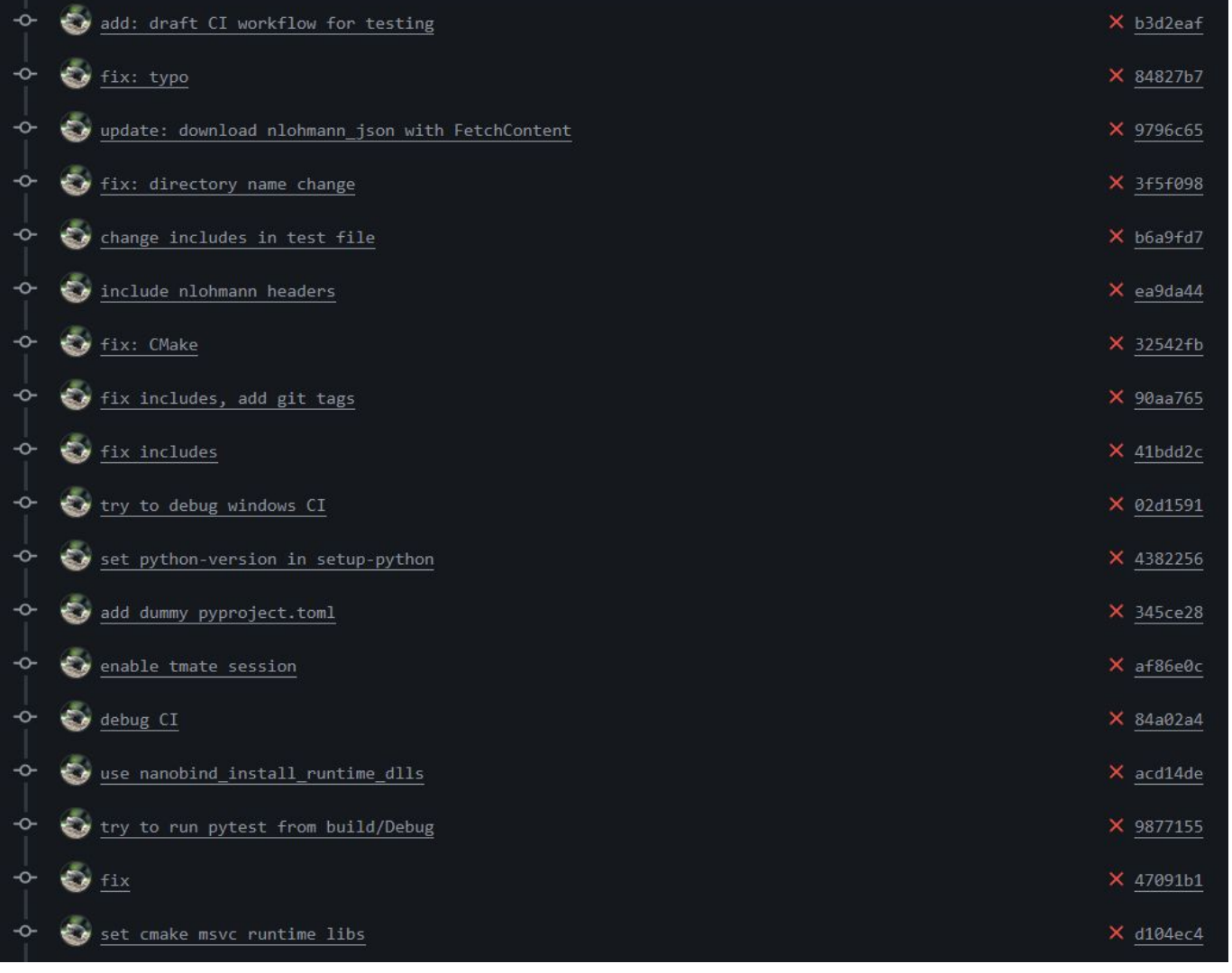
The culprit revealed

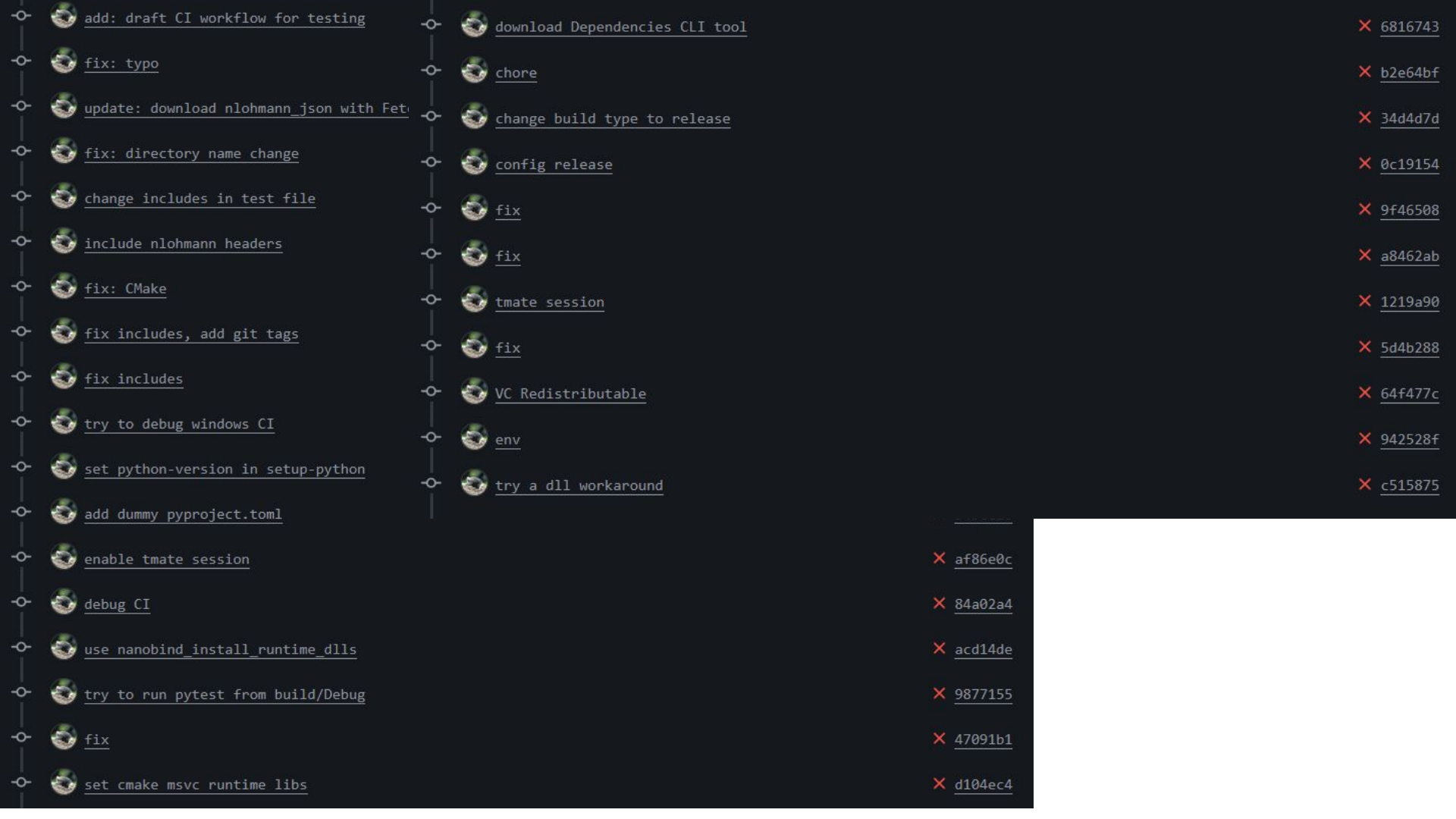
```
bool from_python(handle src, uint8_t flags, cleanup_list *cleanup) {
    try {
        value = nbjson::to_json<nl::json>(src);
        return true;
    }
    catch (nbjson::CircularReferenceError &e) {
        throw e;
    }
    catch (...) {
        return false;
    }
}
```

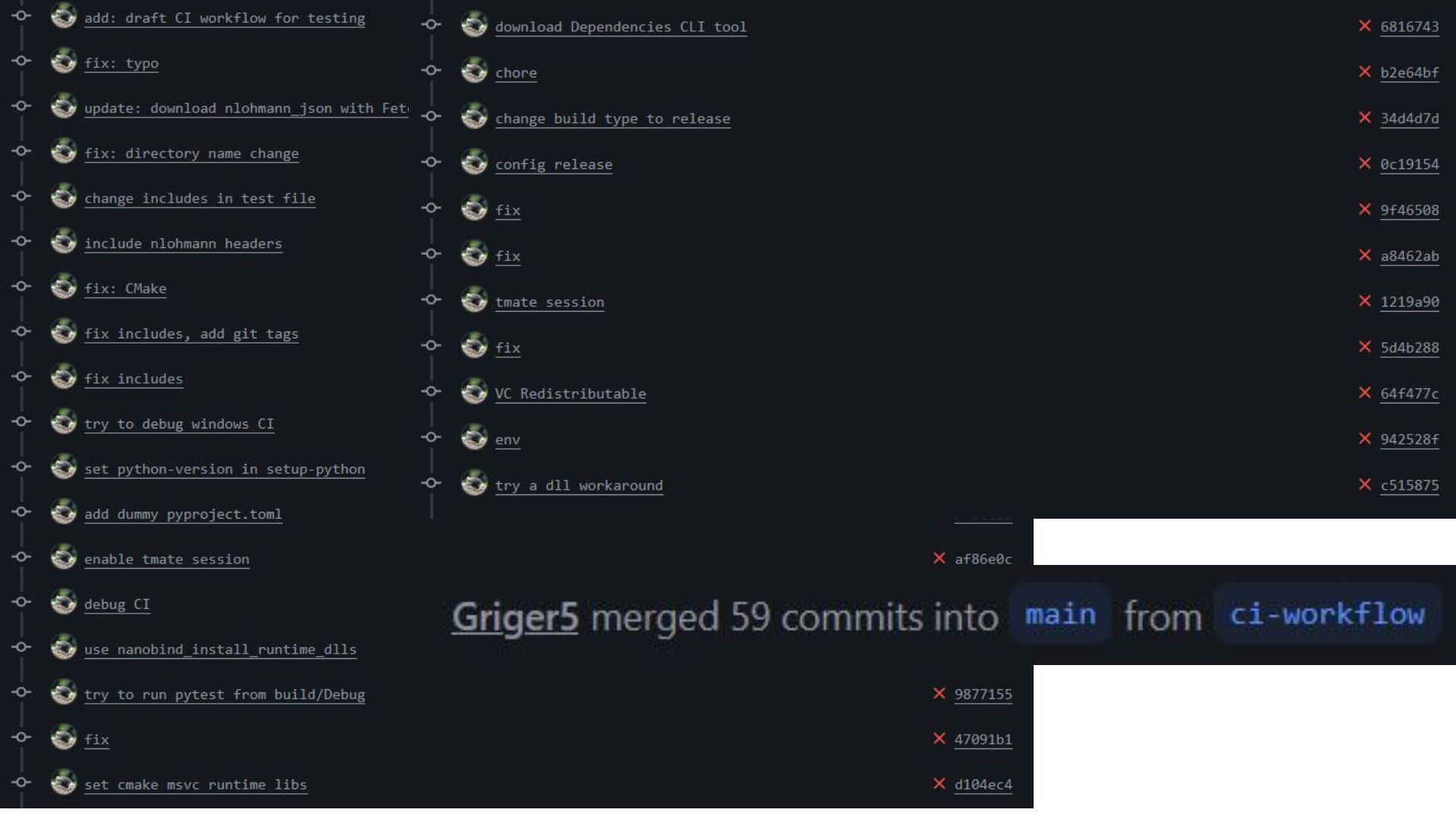
Let’s add CI testing on Github Actions!

Let’s add CI testing on Github Actions!

```yaml
steps:
- uses: actions/checkout@v4

- name: Setup Python ${{ matrix.python }}
  uses: actions/setup-python@v5
  with:
    python-version: ${{ matrix.python }}
    cache: 'pip'

- name: Install pytest
  run: python -m pip install pytest

- name: Configure CMake
  run: cmake -S $GITHUB_WORKSPACE -B $GITHUB_WORKSPACE/build/ -DBUILD_TESTS=1

- name: Build test modules
  run: cmake --build $GITHUB_WORKSPACE/build/

- name: Run tests
  run: |
    cd build
    python -m pytest ../tests
```

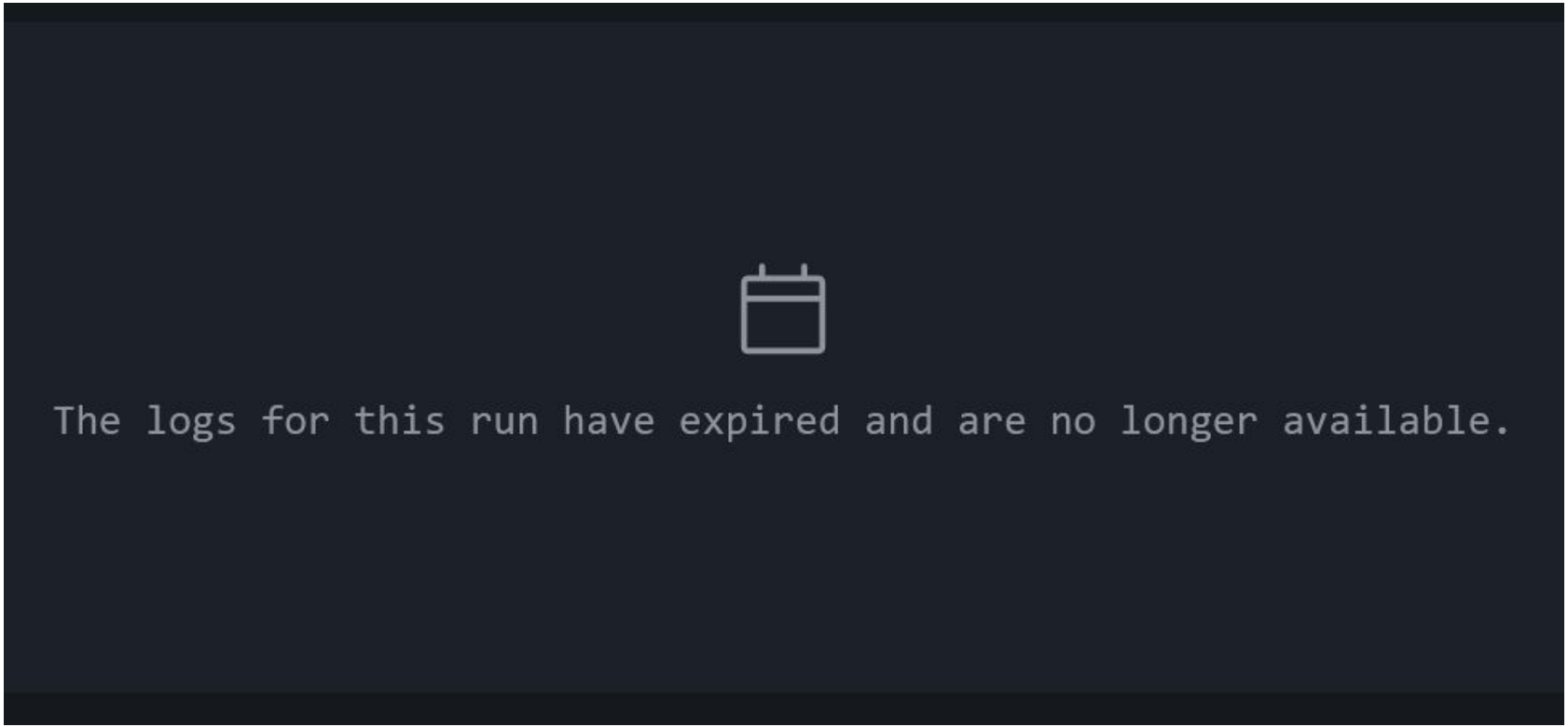

add: draft CI workflow for testing
b3d2eaf
fix: typo
84827b7
update: download nlohmann_json with FetchContent
9796c65
fix: directory name change
3f5f098
change includes in test file
b6a9fd7
include nlohmann headers
ea9da44
fix: CMake
32542fb
fix includes, add git tags
90aa765
fix includes
41bdd2c
try to debug windows CI
02d1591
set python-version in setup-python
4382256
add dummy pyproject.toml
345ce28
enable tmate session
af86e0c
debug CI
84a02a4
use nanobind_install_runtime_dlls
acd14de
try to run pytest from build/Debug
9877155
fix
47091b1
set cmake msvc runtime libs
d104ec4


add: draft CI workflow for testing
fix: typo
update: download nlohmann_json with Fet
fix: directory name change
change includes in test file
include nlohmann headers
fix: CMake
fix includes, add git tags
fix includes
try to debug windows CI
set python-version in setup-python
add dummy pyproject.toml
enable tmate session
af86e0c
debug CI
84a02a4
use nanobind_install_runtime_dlls
acd14de
try to run pytest from build/Debug
9877155
fix
47091b1
set cmake msvc runtime libs
d104ec4
download Dependencies CLI tool
6816743
chore
b2e64bf
change build type to release
34d4d7d
config release
0c19154
fix
9f46508
fix
a8462ab
tmate session
1219a90
fix
5d4b288
VC Redistributable
64f477c
env
942528f
try a dll workaround
c515875


add: draft CI workflow for testing
fix: typo
update: download nlohmann_json with Fet
fix: directory name change
change includes in test file
include nlohmann headers
fix: CMake
fix includes, add git tags
fix includes
try to debug windows CI
set python-version in setup-python
add dummy pyproject.toml
enable tmate session
debug CI
use nanobind_install_runtime_dlls
try to run pytest from build/Debug
fix
set cmake msvc runtime libs
download Dependencies CLI tool
chore
change build type to release
config release
fix
fix
tmate session
fix
VC Redistributable
env
try a dll workaround
6816743
b2e64bf
34d4d7d
0c19154
9f46508
a8462ab
1219a90
5d4b288
64f477c
942528f
c515875
af86e0c
9877155
47091b1
d104ec4
Griger5 merged 59 commits into main from ci-workflow

Let’s look at some failing workflow runs

Let’s look at some failing workflow runs



- Build and run tests on ubuntu-latest
- Build and run tests on ubuntu-latest
- Build and run tests on ubuntu-latest
- Build and run tests on ubuntu-latest
- Build and run tests on macos-latest
- Build and run tests on macos-latest
- Build and run tests on macos-latest
- Build and run tests on macos-latest
- Build and run tests on windows-latest
- Build and run tests on windows-latest
- Build and run tests on windows-latest
- Build and run tests on windows-latest

What were the problems?

What were the problems?

- Wrong yaml syntax

What were the problems?

- Wrong yaml syntax
- Wrong paths (specifically Windows because of MSVC)

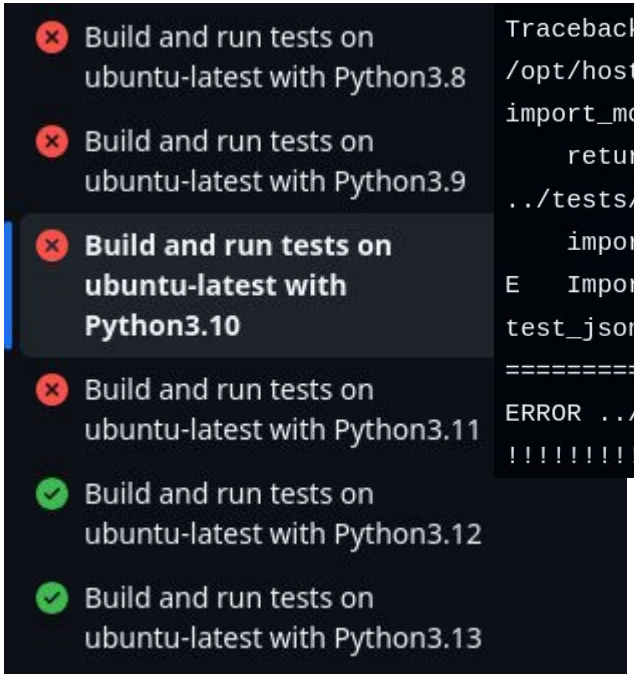
What were the problems?

- Wrong yaml syntax
- Wrong paths (specifically Windows because of MSVC)
- Python package not found (screwed up installation)

What were the problems?

- Wrong yaml syntax
- Wrong paths (specifically Windows because of MSVC)
- Python package not found (screwed up installation)
- But what took most of my time was…

A head-scratching time waster




```
Traceback:
/opt/hostedtoolcache/Python/3.10.20/x64/lib/python3.10/importlib/__init__.py:126: in
import_module
    return _bootstrap._gcd_import(name[level:], package, level)
../tests/test_json.py:5: in <module>
    import test_json_ext as t
E   ImportError: /home/runner/work/nanobind_json/nanobind_json/build/
test_json_ext.cpython-310-x86_64-linux-gnu.so: undefined symbol: PyObject_Vectorcall
=========================== short test summary info ============================
ERROR ../tests/test_json.py
!!!!!!!!!!!!!!!!!!! Interrupted: 1 error during collection !!!!!!!!!!!!!!!!!!!
```

A head-scratching time waster

- Build and run tests on macos-latest with Python3.8
- Build and run tests on macos-latest with Python3.9
- Build and run tests on macos-latest with Python3.10
- **Build and run tests on macos-latest with Python3.11**
- Build and run tests on macos-latest with Python3.12
- Build and run tests on macos-latest with Python3.13

```
Traceback:
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/importlib/
__init__.py:126: in import_module
    return _bootstrap._gcd_import(name[level:], package, level)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
../tests/test_json.py:5: in <module>
    import test_json_ext as t
E   ImportError: dlopen(/Users/runner/work/nanobind_json/nanobind_json/build/
test_json_ext.cpython-311-darwin.so, 0x0002): symbol not found in flat namespace
'_PyDict_GetItemRef'
=========================== short test summary info ============================
ERROR ../tests/test_json.py
!!!!!!!!!!!!!!!!!!! Interrupted: 1 error during collection !!!!!!!!!!!!!!!!!!!
```

A head-scratching time waster

- ✓ Build and run tests on windows-latest with Python3.8
- ✓ Build and run tests on windows-latest with Python3.9
- ✓ Build and run tests on windows-latest with Python3.10
- ✓ Build and run tests on windows-latest with Python3.11
- ✓ Build and run tests on windows-latest with Python3.12
- ✗ Build and run tests on windows-latest with Python3.13

```
Traceback:
C:\hostedtoolcache\windows\Python\3.13.13\x64\Lib\importlib\__init__.py:88: in
import_module
    return _bootstrap._gcd_import(name[level:], package, level)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
..\..\tests\test_json.py:5: in <module>
    import test_json_ext as t
E   ImportError: DLL load failed while importing test_json_ext: The specified module could
not be found.
=========================== short test summary info ===========================
ERROR ..\..\tests\test_json.py
!!!!!!!!!!!!!!!!!!!! Interrupted: 1 error during collection !!!!!!!!!!!!!!!!!!!!
```

The second culprit revealed

The second culprit revealed

```
-- The C compiler identification is MSVC 19.44.35226.0
-- The CXX compiler identification is MSVC 19.44.35226.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/Program Files/Microsoft Visual Studio/2022/Enterprise/VC/Tools/
MSVC/14.44.35207/bin/Hostx64/x64/cl.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: C:/Program Files/Microsoft Visual Studio/2022/Enterprise/VC/Tools/
MSVC/14.44.35207/bin/Hostx64/x64/cl.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found Python: C:/hostedtoolcache/windows/Python/3.13.13/x64/python3.exe (found suitable version "3.13.13",
minimum required is "3.9") found components: Interpreter Development.Module Development.SABIModule
-- Using the multi-header code from D:/a/nanobind_json/nanobind_json/build/_deps/nlohmann_json-src/include/
-- Found Python: C:/hostedtoolcache/windows/Python/3.14.4/x64/python3.exe (found version "3.14.4") found
components: Interpreter Development Development.Module Development.Embed
-- Configuring done (32.3s)
-- Generating done (0.1s)
-- Build files have been written to: D:/a/nanobind_json/nanobind_json/build
```

The second culprit revealed

```
-- The C compiler identification is MSVC 19.44.35226.0
-- The CXX compiler identification is MSVC 19.44.35226.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/Program Files/Microsoft Visual Studio/2022/Enterprise/VC/Tools/
MSVC/14.44.35207/bin/Hostx64/x64/cl.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: C:/Program Files/Microsoft Visual Studio/2022/Enterprise/VC/Tools/
MSVC/14.44.35207/bin/Hostx64/x64/cl.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found Python: C:/hostedtoolcache/windows/Python/3.13.13/x64/python3.exe (found suitable version "3.13.13",
minimum required is "3.9") found components: Interpreter Development.Module Development.SABIModule
-- Using the multi-header code from D:/a/nanobind_json/nanobind_json/build/_deps/nlohmann_json-src/include/
-- Found Python: C:/hostedtoolcache/windows/Python/3.14.4/x64/python3.exe (found version "3.14.4") found
components: Interpreter Development Development.Module Development.Embed
-- Configuring done (32.3s)
-- Generating done (0.1s)
-- Build files have been written to: D:/a/nanobind_json/nanobind_json/build
```

The second culprit revealed

```
-- The C compiler identification is MSVC 19.44.35226.0
-- The CXX compiler identification is MSVC 19.44.35226.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/Program Files/Microsoft Visual Studio/2022/Enterprise/VC/Tools/
MSVC/14.44.35207/bin/Hostx64/x64/cl.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: C:/Program Files/Microsoft Visual Studio/2022/Enterprise/VC/Tools/
MSVC/14.44.35207/bin/Hostx64/x64/cl.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found Python: C:/hostedtoolcache/windows/Python/3.13.13/x64/python3.exe (found suitable version "3.13.13",
minimum required is "3.9") found components: Interpreter Development.Module Development.SABIModule
-- Using the multi-header code from D:/a/nanobind_json/nanobind_json/build/_deps/nlohmann_json-src/include/
-- Found Python: C:/hostedtoolcache/windows/Python/3.14.4/x64/python3.exe (found version "3.14.4") found
components: Interpreter Development Development.Module Development.Embed
-- Configuring done (32.3s)
-- Generating done (0.1s)
-- Build files have been written to: D:/a/nanobind_json/nanobind_json/build
```

The second culprit revealed

```
-- The C compiler identification is MSVC 19.44.35226.0
-- The CXX compiler identification is MSVC 19.44.35226.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/Program Files/Microsoft Visual Studio/2022/Enterprise/VC/Tools/
MSVC/14.44.35207/bin/Hostx64/x64/cl.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: C:/Program Files/Microsoft Visual Studio/2022/Enterprise/VC/Tools/
MSVC/14.44.35207/bin/Hostx64/x64/cl.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found Python: C:/hostedtoolcache/windows/Python/3.13.13/x64/python3.exe (found suitable version "3.13.13",
minimum required is "3.9") found components: Interpreter Development.Module Development.Embed
-- Fetching the nlohmann_json single-header code from D:/a/nanobind_json/nanobind_json/build/_deps/nlohmann_json-src/include/
-- Found Python: C:/hostedtoolcache/windows/Python/3.14.4/x64/python3.exe (found version "3.14.4") found
components: Interpreter Development Development.Module Development.Embed
-- Configuring done (32.3s)
-- Generating done (0.1s)
-- Build files have been written to: D:/a/nanobind_json/nanobind_json/build
```

The second culprit revealed

```
-- The C compiler identification is MSVC 19.44.35226.0
-- The CXX compiler identification is MSVC 19.44.35226.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/Program Files/Microsoft Visual Studio/2022/Enterprise/VC/Tools/
MSVC/14.44.35207/bin/Hostx64/x64/cl.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: C:/Program Files/Microsoft Visual Studio/2022/Enterprise/VC/Tools/
MSVC/14.44.35207/bin/Hostx64/x64/cl.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found Python: C:/hostedtoolcache/windows/Python/3.13.13/x64/python3.exe (found suitable version "3.13.13",
M                 is "3.9") found components: Interpreter Development.Module
-                 -header code from D:/a/nanobind_json/nanobind_json/build/                 /include/
-- Found Python: C:/hostedtoolcache/windows/Python/3.14.4/x64/python3.exe (found version "3.14.4") found
C                 preter Development Development.Module Development.Embed
-- Configuring done (32.3s)
-- Generating done (0.1s)
-- Build files have been written to: D:/a/nanobind_json/nanobind_json/build
```

The second culprit revealed

```
include(FetchContent)

FetchContent_Declare(
        nanobind
        GIT_REPOSITORY https://github.com/wjakob/nanobind.git
        GIT_TAG master
)

FetchContent_Declare(
        nlohmann_json
        GIT_REPOSITORY https://github.com/nlohmann/json.git
        GIT_TAG master
)

FetchContent_MakeAvailable(nanobind nlohmann_json)

include_directories(build/_deps/nlohmann_json-src/include)

set(nanobind_REQUIRED_VERSION 2.2.0)
set(nlohmann_json_REQUIRED_VERSION 3.2.0)

find_package(Python COMPONENTS Interpreter Development REQUIRED)
```

The Package is Fixed

The Package is Fixed

- Build and run tests on ubuntu-latest with Python 3.9
- Build and run tests on ubuntu-latest with Python 3.10
- Build and run tests on ubuntu-latest with Python 3.11
- Build and run tests on ubuntu-latest with Python 3.12
- Build and run tests on ubuntu-latest with Python 3.13
- Build and run tests on ubuntu-latest with Python 3.14

- Build and run tests on ubuntu-24.04-arm with Python 3.9
- Build and run tests on ubuntu-24.04-arm with Python 3.10
- Build and run tests on ubuntu-24.04-arm with Python 3.11
- Build and run tests on ubuntu-24.04-arm with Python 3.12
- Build and run tests on ubuntu-24.04-arm with Python 3.13
- Build and run tests on ubuntu-24.04-arm with Python 3.14

- Build and run tests on windows-latest with Python 3.9
- Build and run tests on windows-latest with Python 3.10
- Build and run tests on windows-latest with Python 3.11
- Build and run tests on windows-latest with Python 3.12
- Build and run tests on windows-latest with Python 3.13
- Build and run tests on windows-latest with Python 3.14

- Build and run tests on macos-latest with Python 3.9
- Build and run tests on macos-latest with Python 3.10
- Build and run tests on macos-latest with Python 3.11
- Build and run tests on macos-latest with Python 3.12
- Build and run tests on macos-latest with Python 3.13
- Build and run tests on macos-latest with Python 3.14

The Package is Fixed

Hi Gracjan,

this looks great -- could I ask you to make a PR in the nanobind repository to link to your code?

Best,
Wenzel

The Package is Fixed

Hi Gracjan,

this looks great -- could I ask you to make a PR in the nanobind repository to link to your code?

Best,
Wenzel

Is there a way to pass `JSON` objects between Python and C++? ¶

Yes, an unofficial, currently maintained, package supporting that can be found here. It is based on a similar package for `pybind11` called `pybind11_json`.

The Package is Fixed

Hi Gracjan,

this looks great -- could I ask you to make a PR in the nanobind repository to link to your code?

Best,
Wenzel

https://github.com/Griger5/nanobind_json

The Package is Fixed

Ordered JSON support!

```
template <> struct type_caster<nl::ordered_json> {
public:
    NB_TYPE_CASTER(nl::ordered_json, const_name("OrderedJSON"))
```

support for `nlohmann::ordered_json` in addition to `nlohmann::json` #63

Open

slayoo opened on May 18, 2023

Adding support for the `ordered_json` type in addition to currently used `json` would be beneficial for users intending to preserve order of elements in the JSON structures.

Help wanted! Contributions welcome!

Help wanted! Contributions welcome!

- If you find any bugs, feel free to create an issue or a PR!

Help wanted! Contributions welcome!

- If you find any bugs, feel free to create an issue or a PR!
- The library only supports nlohmann/json, it would be awesome to also include: **simdjson**, **Boost.JSON**, **picojson** and others!

Bonus: nanobind crash course!

- Building your first bindings
- Exposing object properties to Python
- STL inter-operability
- Writing your own type caster

nanobind: minimal example (mini_ext.cpp)

```cpp
#include <nanobind/nanobind.h>

namespace nb = nanobind;

NB_MODULE(mini_ext, m) {
    m.doc() = "Minimal nanobind example";

    m.def(
        "add",
        [](int a, int b) { return a + b; },
        nb::arg("a"),
        nb::arg("b"),
        "Add two integers"
    );
}
```

nanobind: CMakeLists.txt

```
cmake_minimum_required(VERSION 3.18...3.30)
project(nanobind_demo LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Python 3.12 COMPONENTS
    Interpreter Development.Module
    REQUIRED)

include(FetchContent)

FetchContent_Declare(
    nanobind
    GIT_REPOSITORY https://github.com/wjakob/nanobind.git
    GIT_TAG v2.5.0
)

FetchContent_MakeAvailable(nanobind)

nanobind_add_module(mini_ext mini_ext.cpp)
nanobind_add_module(counter_ext counter_ext.cpp)
nanobind_add_module(stl_ext stl_ext.cpp)

install(TARGETS mini_ext LIBRARY DESTINATION .)
install(TARGETS counter_ext LIBRARY DESTINATION .)
install(TARGETS stl_ext LIBRARY DESTINATION .)
```

nanobind: pyproject.toml

```
[build-system]
requires = [
    "scikit-build-core >=0.4.3",
    "nanobind >=2.0.0"
]
build-backend = "scikit_build_core.build"

[project]
name = "nanobind_demo"
version = "0.1.0"
```

nanobind: build and use!

```
pip install -e .
```

```
Python 3.12.3 (main, Jun 19 2026, 12:46:00) [GCC 13.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
Ctrl click to launch VS Code Native REPL
>>> import mini_ext as m
>>> m.add(4, 5)
9
```

nanobind: Classes

```cpp
class Counter {
public:
    explicit Counter(int start = 0) : value_(start) {}

    void increment() { ++value_; }

    int value() const { return value_; }

    void set_value(int v) { value_ = v; }

private:
    int value_;
};
```

```cpp
NB_MODULE(counter_ext, m) {
    nb::class_<Counter>(m, "Counter", "A tiny mutable counter class")
        .def(
            nb::init<int>(), nb::arg("start") = 0, "Create a counter"
        )
        .def(
            "increment", &Counter::increment, "Increase the counter by one"
        )
        .def(
            "get", &Counter::value, "Method-style getter"
        )
        .def_prop_ro(
            "value_ro", &Counter::value, "Read-only property"
        )
        .def_prop_rw(
            "value_rw", &Counter::value, &Counter::set_value, "Read-write property"
        )
        .def(
            "__repr__",
            [](const Counter &c) {
                return "Counter(" + std::to_string(c.value()) + ")";
            }
        );
}
```

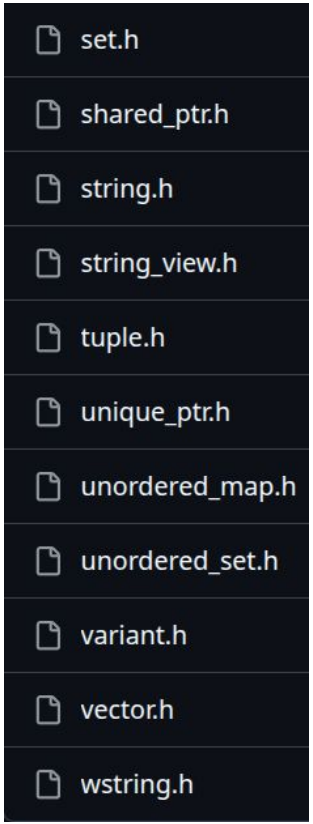
```
Python 3.12.3 (main, Jun 19 2026, 12:46:00) [GCC 13.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
Ctrl click to launch VS Code Native REPL
>>> import counter_ext as c
>>> a = c.Counter()
>>> a.get()
0
>>> a.value_ro
0
>>> a.value_rw
0
>>> a.value_rw = 5
>>> a.value_ro = 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: property of 'Counter' object has no setter
>>> print(a)
Counter(5)
```

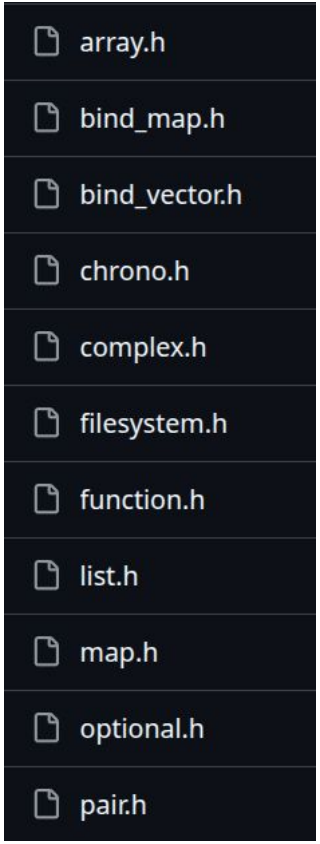
nanobind: STL usage

```
NB_MODULE(stl_ext, m) {
    m.doc() = "STL conversion demo";

    m.def(
        "upper",
        [](std::vector<std::string> words) {
            for (auto &w : words) {
                std::transform(
                    w.begin(),
                    w.end(),
                    w.begin(),
                    [](unsigned char ch) {
                        return static_cast<char>(std::toupper(ch));
                    }
                );
            }
            return words;
        },
        nb::arg("words"),
        "Convert a list of Python strings to a list of uppercase strings"
    );
}
```

nanobind: STL usage

```cpp
NB_MODULE(stl_ext, m) {
    m.doc() = "STL conversion demo";

    m.def(
        "upper",
        [](std::vector<std::string> words) {
            for (auto &w : words) {
                std::transform(
                    w.begin(),
                    w.end(),
                    w.begin(),
                    [](unsigned char ch) {
                        return static_cast<char>(std::toupper(ch));
                    }
                );
            }
            return words;
        },
        nb::arg("words"),
        "Convert a list of Python strings to a list of uppercase strings"
    );
}
```





nanobind: custom type caster

```
template <typename Type> struct type_caster<std::valarray<Type>> {
    NB_TYPE_CASTER(std::valarray<Type>, const_name("[") + const_name("std::valarray") + const_name("]"))

    using Caster = make_caster<Type>;
```

```
static handle from_cpp(const std::valarray<Type> &src, rv_policy policy, cleanup_list *cleanup) {
    nb::list obj;
    for (const auto& elem : src) {
        obj.append(elem);
    }

    return obj.release();
}
```

nanobind: custom type caster

```cpp
bool from_python(handle src, uint8_t flags, cleanup_list *cleanup) noexcept {
    if (nb::isinstance<nb::list>(src)) {
        try {
            auto py_array = nb::cast<nb::list>(src);
            size_t size = py_array.size();

            value.resize(size);

            for (size_t i = 0; i < size; i++) {
                value[i] = nb::cast<Type>(py_array[i]);
            }

            return true;
        }
        catch (...) {
            PyErr_Clear();
            return false;
        }
    }

    else if (nb::isinstance<nb::ndarray<Type, nb::ndim<1>>>(src)) {
        try {
            auto py_array = nb::cast<nb::ndarray<Type, nb::ndim<1>>>(src);
            size_t size = py_array.size();
            auto *data = py_array.data();

            value.resize(size);

            for (size_t i = 0; i < size; i++) {
                value[i] = data[i];
            }

            return true;
        }
        catch (...) {
            PyErr_Clear();
            return false;
        }
    }

    return false;
}
```

```cpp
template <typename JSONType>
inline nb::object from_json(const JSONType &j) {
    if (j.is_null()) {
        return nb::none();
    }
    else if (j.is_boolean()) {
        return nb::bool_(j.template get<bool>());
    }
    else if (j.is_number_integer()) {
        return nb::int_(j.template get<long long>());
    }
    else if (j.is_number_float()) {
        return nb::float_(j.template get<double>());
    }
    else if (j.is_string()) {
        return nb::str(j.template get<std::string>().c_str());
    }
    else if (j.is_array()) {
        nb::list obj;
        for (const auto &el : j) {
            obj.append(from_json<JSONType>(el));
        }
        return obj;
    }
    else {
        nb::dict obj;
        for (typename JSONType::const_iterator it = j.cbegin(); it != j.cend(); ++it) {
            obj[nb::str(it.key().c_str())] = from_json<JSONType>(it.value());
        }
        return obj;
    }
}
```

```cpp
template <typename JSONType>
inline JSONType to_json(const nb::handle &obj, std::set<const PyObject *> &refs) {
    if (obj.ptr() == nullptr || obj.is_none()) {
        return nullptr;
    }
    else if (nb::isinstance<nb::bool_>(obj)) {
        return nb::cast<bool>(obj);
    }
    else if (nb::isinstance<nb::int_>(obj)) {
        return nb::cast<long long>(obj);
    }
    else if (nb::isinstance<nb::float_>(obj)) {
        return nb::cast<double>(obj);
    }
    else if (nb::isinstance<nb::bytes>(obj)) {
        nb::module_ base64 = nb::module_::import_("base64");
        return nb::cast<std::string>(base64.attr("b64encode")(obj).attr("decode")("utf-8"));
    }
    else if (nb::isinstance<nb::str>(obj)) {
        return nb::cast<std::string>(obj);
    }
```

```
else if (nb::isinstance<nb::tuple>(obj) || nb::isinstance<nb::list>(obj)) {
    auto insert_ret = refs.insert(obj.ptr());
    if (!insert_ret.second) {
        throw CircularReferenceError("Circular reference detected");
    }

    auto out = JSONType::array();
    for (const nb::handle value : obj) {
        out.push_back(to_json<JSONType>(value, refs));
    }

    refs.erase(insert_ret.first);

    return out;
}
else if (nb::isinstance<nb::dict>(obj)) {
    auto insert_ret = refs.insert(obj.ptr());
    if (!insert_ret.second) {
        throw CircularReferenceError("Circular reference detected");
    }

    auto out = JSONType::object();
    for (const nb::handle key : obj) {
        out[nb::cast<std::string>(nb::str(key))] = to_json<JSONType>(obj[key], refs);
    }

    refs.erase(insert_ret.first);

    return out;
}
else {
    throw std::runtime_error("to_json not implemented for this type of object: " + nb::cast<std::string>(nb::repr(obj)));
}
```

```
else if (nb::isinstance<nb::tuple>(obj) || nb::isinstance<nb::list>(obj)) {
    auto insert_ret = refs.insert(obj.ptr());
    if (!insert_ret.second) {
        throw CircularReferenceError("Circular reference detected");
    }

    auto out = JSONType::array();
    for (const nb::handle value : obj) {
        out.push_back(to_json<JSONType>(value, refs));
    }

    refs.erase(insert_ret.first);

    return out;
}
else if (nb::isinstance<nb::dict>(obj)) {
    auto insert_ret = refs.insert(obj.ptr());
    if (!insert_ret.second) {
        throw CircularReferenceError("Circular reference detected");
    }

    auto out = JSONType::object();
    for (const nb::handle key : obj) {
        out[nb::cast<std::string>(nb::str(key))] = to_json<JSONType>(obj[key], refs);
    }

    refs.erase(insert_ret.first);

    return out;
}
else {
    throw std::runtime_error("to_json not implemented for this type of object: " + nb::cast<std::string>(nb::repr(obj)));
}
```

```
template <typename JSONType>
inline JSONType to_json(const nb::handle &obj) {
    std::set<const PyObject *> refs;
    return to_json<JSONType>(obj, refs);
}
```

```
namespace nlohmann {
    #define MAKE_NLJSON_SERIALIZER_DESERIALIZER(T)                  \
    template <>                                                     \
    struct adl_serializer<T> {                                      \
        inline static void to_json(json &j, const T &obj) {         \
            j = nbjson::to_json<json>(obj);                         \
        }                                                           \
                                                                    \
        inline static T from_json(const json &j) {                  \
            return nb::cast<T>(nbjson::from_json<json>(j));         \
        }                                                           \
                                                                    \
        inline static void to_json(ordered_json &j, const T &obj) { \
            j = nbjson::to_json<ordered_json>(obj);                 \
        }                                                           \
                                                                    \
        inline static T from_json(const ordered_json &j) {          \
            return nb::cast<T>(nbjson::from_json<ordered_json>(j)); \
        }                                                           \
    }

    #define MAKE_NLJSON_SERIALIZER_ONLY(T)                          \
    template <>                                                     \
    struct adl_serializer<T> {                                      \
        inline static void to_json(json &j, const T &obj) {         \
            j = nbjson::to_json<json>(obj);                         \
        }                                                           \
                                                                    \
        inline static void to_json(ordered_json &j, const T &obj) { \
            j = nbjson::to_json<ordered_json>(obj);                 \
        }                                                           \
    }
```

```
namespace nlohmann {
    #define MAKE_NLJSON_SERIALIZER_DESERIALIZER(T)                    \
    template <>                                                       \
    struct adl_serializer<T> {                                        \
        inline static void to_json(json &j, const T &obj) {           \
            j = nbjson::to_json<json>(obj);                           \
        }                                                             \
                                                                      \
        inline static T from_json(const json &j) {                    \
            return nb::cast<T>(nbjson::from_json<json>(j));           \
        }                                                             \
                                                                      \
        inline static void to_json(ordered_json &j, const T &obj) {   \
            j = nbjson::to_json<ordered_json>(obj);                   \
        }                                                             \
                                                                      \
        inline static T from_json(const ordered_json &j) {            \
            return nb::cast<T>(nbjson::from_json<ordered_json>(j));   \
        }                                                             \
    }

    #define MAKE_NLJSON_SERIALIZER_ONLY(T)                            \
    template <>                                                       \
    struct adl_serializer<T> {                                        \
        inline static void to_json(json &j, const T &obj) {           \
            j = nbjson::to_json<json>(obj);                           \
        }                                                             \
                                                                      \
        inline static void to_json(ordered_json &j, const T &obj) {   \
            j = nbjson::to_json<ordered_json>(obj);                   \
        }                                                             \
    }
```

```
MAKE_NLJSON_SERIALIZER_DESERIALIZER(nb::object);

MAKE_NLJSON_SERIALIZER_DESERIALIZER(nb::bool_);
MAKE_NLJSON_SERIALIZER_DESERIALIZER(nb::int_);
MAKE_NLJSON_SERIALIZER_DESERIALIZER(nb::float_);
MAKE_NLJSON_SERIALIZER_DESERIALIZER(nb::str);

MAKE_NLJSON_SERIALIZER_DESERIALIZER(nb::list);
MAKE_NLJSON_SERIALIZER_DESERIALIZER(nb::tuple);
MAKE_NLJSON_SERIALIZER_DESERIALIZER(nb::dict);

MAKE_NLJSON_SERIALIZER_ONLY(nb::handle);

#undef MAKE_NLJSON_SERIALIZER_DESERIALIZER
#undef MAKE_NLJSON_SERIALIZER_ONLY
```

The caster is ready!

```cpp
template <> struct type_caster<nl::json> {
public:
    NB_TYPE_CASTER(nl::json, const_name("JSON"))

    bool from_python(handle src, uint8_t flags, cleanup_list *cleanup) {
        try {
            value = nbjson::to_json<nl::json>(src);
            return true;
        }
        catch (nbjson::CircularReferenceError &e) {
            throw e;
        }
        catch (...) {
            return false;
        }
    }

    static handle from_cpp(const nl::json &src, rv_policy policy, cleanup_list *cleanup) noexcept {
        object obj = nbjson::from_json<nl::json>(src);
        return obj.release();
    }
};
```

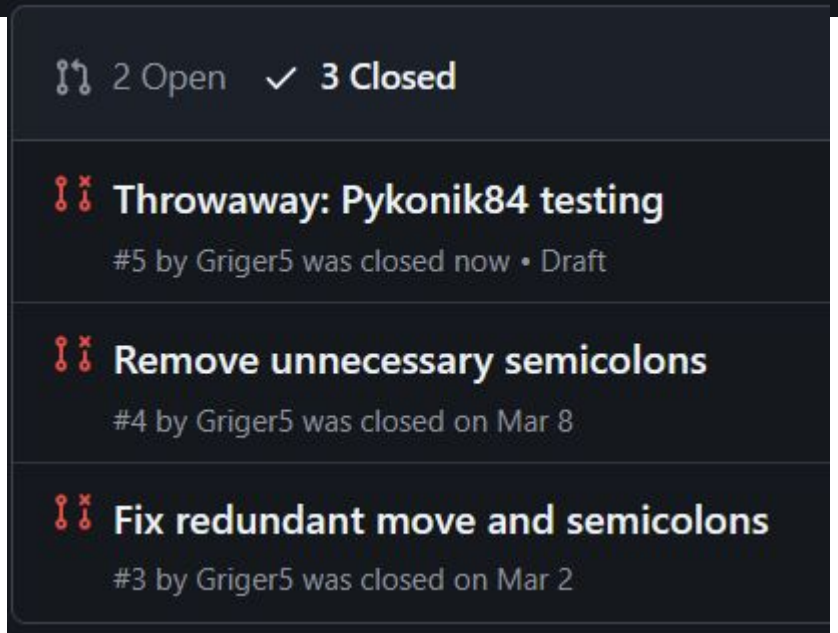
Fun facts behind the scenes

Fun facts behind the scenes

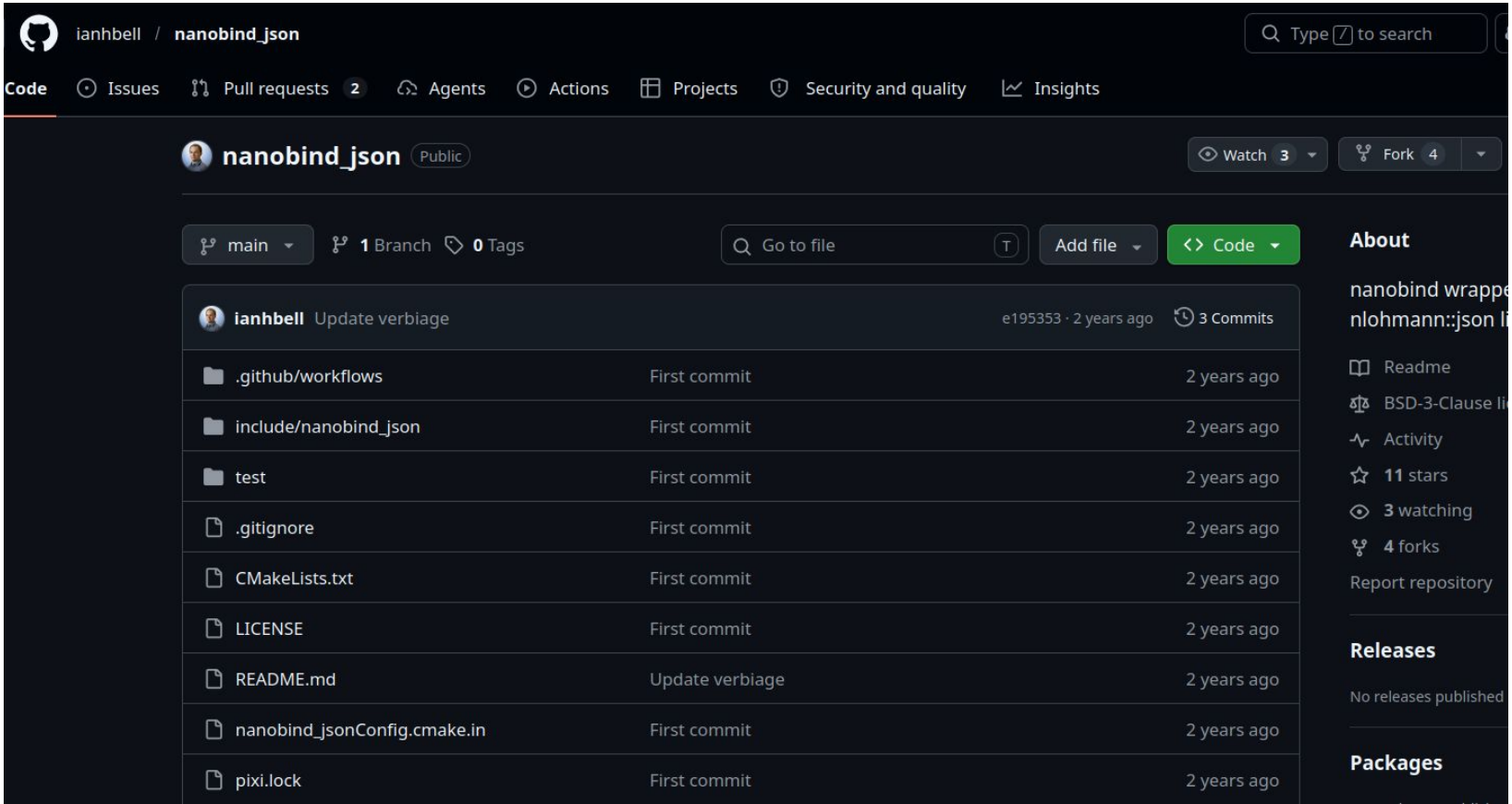
```yaml
# TODO: Run on other Python versions on Windows (Missing/not-matching DLLs error)
- name: Run tests
  if: ${{ startsWith(matrix.os, 'windows') }}
  run: |
    cd build/Release
    python -m pytest ../../tests
```

Fun facts behind the scenes

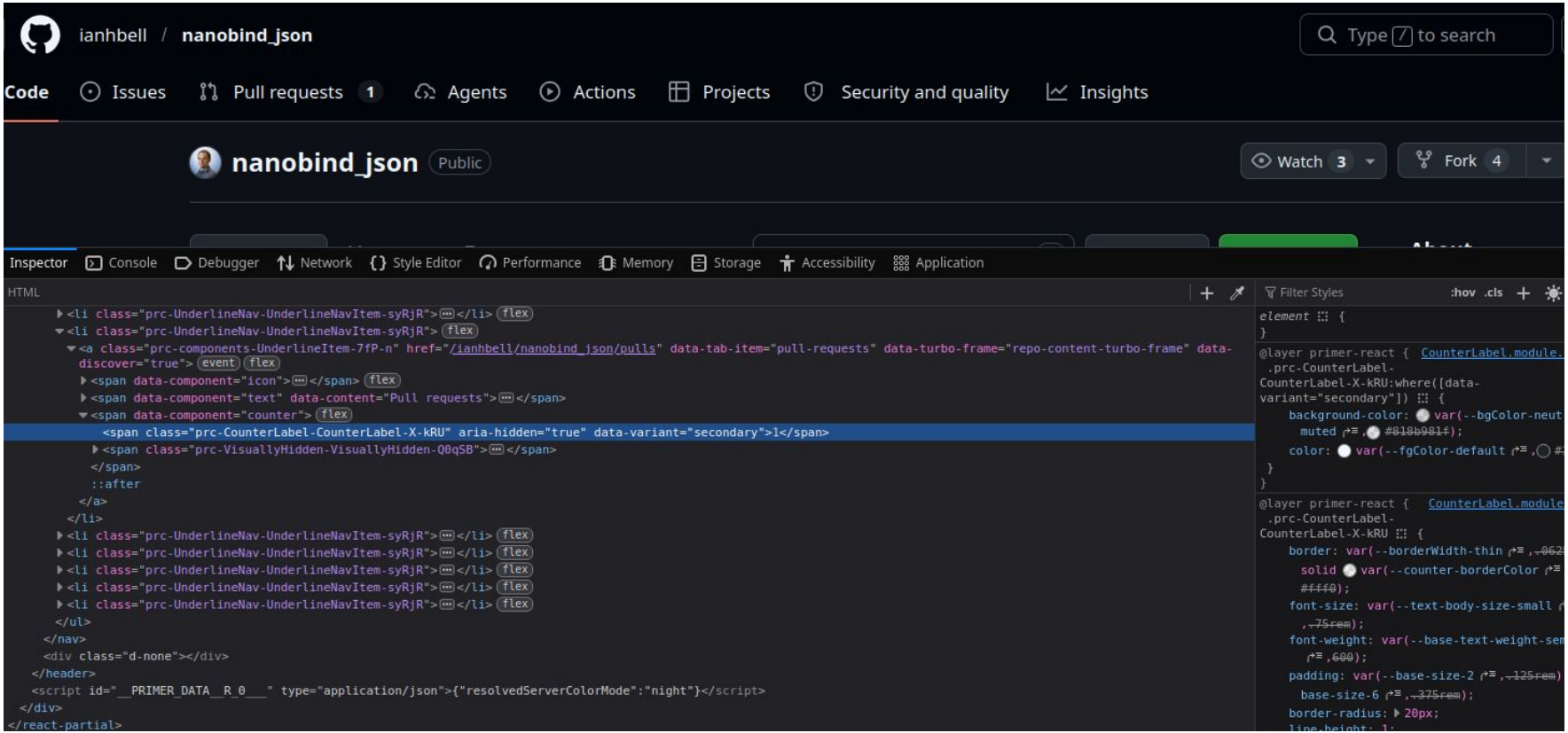
Fun facts behind the scenes



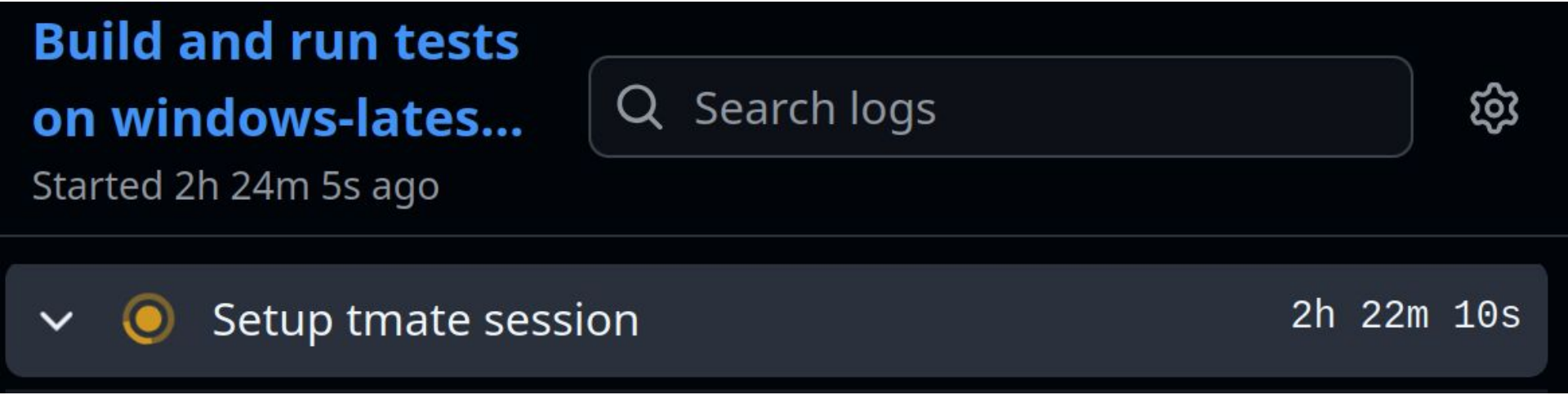
Fun facts behind the scenes



Fun facts behind the scenes



Fun facts behind the scenes



Fun facts behind the scenes

- My first time actually using git cherry-pick!
- I wanted to quickly present the library as a Lightning Talk on Pykonik 79, my laptop decided to not cooperate with HDMI

Thank you!