# What’s new in gstlearn – Software Engineering

## Séminaire Géolearning

Pierre Guillou

Fréjus, le 3 avril 2025

Mines Paris, PSL Research Université



# Plan

1. GitHub CI/CD
2. Python Packaging
3. Serialization
4. C++ Standard Upgrade
5. Performance improvements
6. Debugging with Address Sanitizer
7. Conclusion

# GitHub CI/CD

# Continuous Integration / Continuous Delivery with GitHub Actions

- build gstlearn & execute tests every time a change is proposed
  - on different platforms : Ubuntu, macOS, Windows
  - $\sim$400 tests (C++, R, Python)
- ensure there is no (visible) regression
- also build Python, R packages for releases
  - required dependencies $\Longrightarrow$ static libraries
- always evolving environment $\Longrightarrow$ complex maintenance work

# CI/CD improvements

## Upgrading the workflows

- Latest macOS, Python
- Testing before publishing packages
- Use several packaging systems

| | |
|---|---|
| Ubuntu | apt, pip (pypi) |
| macOS | brew, pip (pypi) |
| Windows | vcpkg, pixi (conda), MSYS pacman |

## Caching binary objects

- `ccache` / `sccache` used with GitHub cache
  - store binary object files (source code, compile command)
  - $\neq$ partial compilation
- faster to test results

# Python Packaging

# Switching to pyproject.toml


- evolutions in Python packaging metadata format these last years
- switching from `setup.py` to `pyproject.toml`
  - `setup.py` is imperative
  - `pyproject.toml` is declarative, more generic
  - `pyproject.toml` supports more tools


## NumPy 2.0 released in June 2024

- a compile-time and run-time dependency of gstlearn
- 2.0 has breaking changes incompatible with 1.*
  - building gstlearn Python packages with 1.* won't work with 2.0
  - however, gstlearn packages built with 2.0 are compatible with 1.*
- a few tries to get the compatible NumPy versions
  - `"numpy>=1.24,<@Python3_NumPy_NEXT_MAJOR@"`

# Python package optional dependencies

- the gstlearn Python package has a lot of dependencies
  NumPy scipy matplotlib pandas
  plotly shapely geopandas
- $\sim$100 MB dependencies to download
- only NumPy is required, everything else is optional
- optional dependency groups have been created
  - `plot` : geopandas, matplotlib, plotly and shapely
  - `conv` : pandas and scipy
  - `pip install gstlearn` : only NumPy
  - to download everything : `pip install gstlearn[all]`
- now : gstlearn + NumPy $\Longrightarrow$ 20 MB

# Serialization

# Serialization

**Serialization** Object in memory $\longrightarrow$ File

**Deserialization** File $\longrightarrow$ object in memory

gstlearn : Neutral Files

- text serialization
- quite slow
- hard to evolve
  - any change in the format makes old files invalid

# Binary serialization with HDF5

## Hierarchical Data Format

- file format to store scientific datasets
- C library with C++ interface
- binary storage
- self-documented

# HDF5 PoC in gstlearn

## HDF5 vs Neutral Files

- for now only implemented for `DbGrid`
- x30 faster than Neutral Files
- TODO
  - references : data structures sharing pointers
  - compression ? binary files can be larger than text ones
  - implement for all other data structures @NDesassis

# C++ Standard Upgrade

# New C++ features

## C++ standards

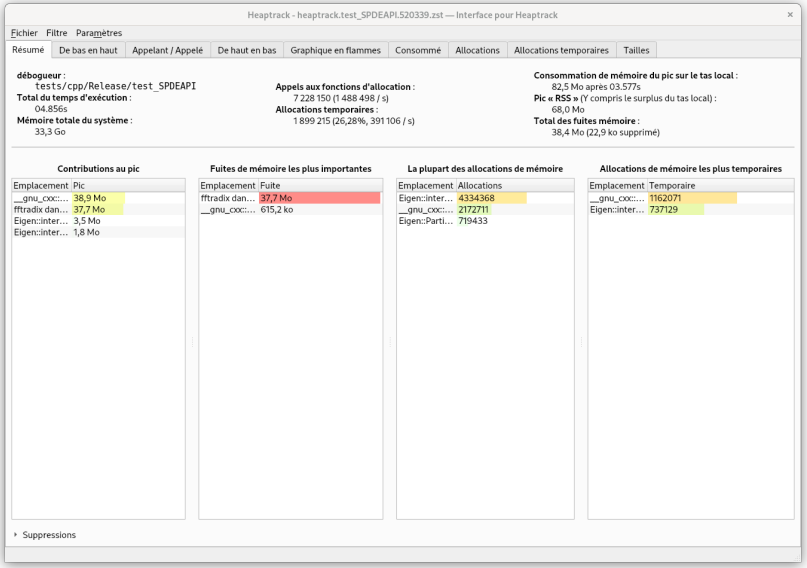
- A new C++ standard every 3 years : 2011, 2014, 2017, 2020...
- New standards bring new stuff
- Old standards might become unsupported by dependencies
- New-ish standards only supported by new-ish compilers...

## gstlearn : Switch from C++11 to C++20

- Filesystem Library
- `std::span`, `std::string_view`
- `std::optional`, `std::variant`

# Performance improvements

# Analyze memory allocations with Heaptrack



# Results

| gstlearn | 1.3.2 (July 2024) | dev (April 2025) |
|---|---|---|
| `test_SPDEAPI` | | |
| Exec time (s) | 16.1 | 5.8 |
| #Allocs | 86 706 557 | 7 230 865 |
| `bench_Db` | | |
| Exec time (s) | 0.4 | 0.2 |
| #Allocs | 6 003 667 | 3 269 |
| `bench_Kriging3DU` | | |
| Exec time (s) | 8.2 | 10.8 |
| #Allocs | 29 007 986 | 24 004 051 |
| `bench_Tree` | | |
| Exec time (s) | 99 | 17 |

# Debugging with Address Sanitizer

# Concept

## ASAN

- A compiler extension
- Helps identify memory bugs
  - out-of-bound accesses
  - ”use-after-free”
  - ”double free”
- Use the `-fsanitize=address` compiler/linker flag
  - debug info provide line numbers : `-g`

# Application

## ASAN for gstlearn

- CMake option to build the library
- One CI job runs tests with it
- Helps debugging
- Other available sanitizers
  - LeakSanitizer
  - ThreadSanitizer

# Conclusion

# Conclusion

## End-user improvements

- Smaller Python packages
- Compatibility with NumPy >= 2.0
- Faster serialization
- Better overall performance
- Robustness

## Developer Experience

- GitHub Actions workflows
- Debugging & profiling Tools

## Perspectives

- Parallelism ?
- Coverage ?

Questions?

# Example : A use-after-free

```cpp
#include <iostream>
#include <vector>

int main(void) {
  std::vector<int> a(2);     // allocate a int buffer
  int *b = a.data();         // take a pointer to the buffer
  a.resize(100);             // realloc the buffer -> b is invalid
  std::cout << b[0] << '\n'; // dereference b -> use-after-free
  return 0;
}
```

# Example : ASAN output

```
==38392==ERROR: AddressSanitizer: heap-use-after-free
READ of size 4
    #0 0x5bc832a683bf in main uaf.cpp:8

located 0 bytes inside of 8-byte region
freed by thread T0 here:
    #0 ...
    #6 0x5bc832a68385 in main uaf.cpp:7

previously allocated by thread T0 here:
    #0 ...
    #7 0x5bc832a6833f in main uaf.cpp:5
```

# Alternative : NetCDF 4


- competitor binary format
- based on HDF5
- C library with C++ interface
  - same complexity than HDF5
  - no references
- harder to build
  - need HDF5 + dependencies, NetCDF C and NetCDF C++ wrapper