Teaching Automatic Differentiation with Interactive C++ Jupyter Notebooks

Aaron Jomy[1], *Vassil Vassilev*[2]

[1]*CERN*, [2]*Princeton University*,

28th EuroAD Workshop

10-12-2025

Motivation

High-level interactive programming languages

like Python simplify differentiable programming

No long edit-compile-run cycles

Simple language syntax

Easy to teach

Jupyter Notebooks are great!

```
[ ]: import jax.numpy as jnp
     from jax import grad, jit, vmap
     from jax import random

     key = random.key(0)
```

```
[6]: grad_tanh = grad(jnp.tanh)
     print(grad_tanh(2.0))
     print(grad(grad(grad(jnp.tanh)))(2.0))
```

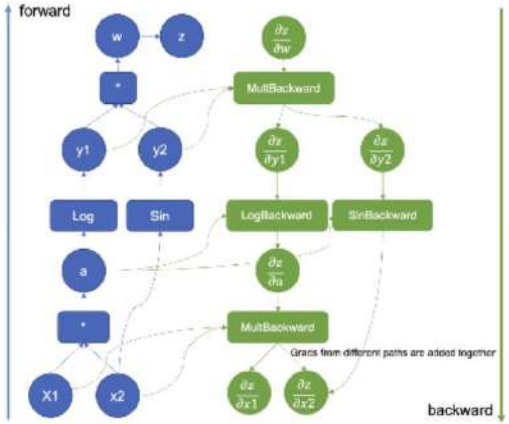
```
0.070650816
0.25265485
```

```
[4]: def sigmoid(x):
         return 0.5 * (jnp.tanh(x / 2) + 1)

     # Outputs probability of a label being true.
     def predict(W, b, inputs):
         return sigmoid(jnp.dot(inputs, W) + b)

     # Build a toy dataset.
     inputs = jnp.array([[0.52, 1.12,  0.77],
                         [0.88, -1.08, 0.15],
                         [0.52, 0.06, -1.30],
                         [0.74, -2.49, 1.39]])
     targets = jnp.array([True, True, False, True])

     # Training loss is the negative log-likelihood of the training examples.
     def loss(W, b):
         preds = predict(W, b, inputs)
         label_probs = preds * targets + (1 - preds) * (1 - targets)
         return -jnp.sum(jnp.log(label_probs))

     # Initial
     key, W_ke
     W = rando
     b = rando
```

```
[5]: # Differentiate `loss` with respect to the first positional argument:
     W_grad = grad(loss, argnums=0)(W, b)
     print('W_grad', W_grad)

     # Since argnums=0 is the default, this does the same thing:
     W_grad = grad(loss)(W, b)
     print('W_grad', W_grad)

     # But we can choose different values too, and drop the keyword:
     b_grad = grad(loss, 1)(W, b)
     print('b_grad', b_grad)

     # Including tuple values
     W_grad, b_grad = grad(loss, (0, 1))(W, b)
     print('W_grad', W_grad)
     print('b_grad', b_grad)
```

```
W_grad [-0.433146  -0.7354605 -1.2598922]
W_grad [-0.433146  -0.7354605 -1.2598922]
b_grad -0.69001776
W_grad [-0.433146  -0.7354605 -1.2598922]
b_grad -0.69001776
```

```
[1]: import torch
```




```
[2]: x = torch.tensor([0.5, 0.75], requires_grad=True)
     y = torch.log(x[0] * x[1]) * torch.sin(x[1])
     y.backward()
     x.grad
```

```
[2]: tensor([1.3633, 0.1912])
```

Sources:
https://docs.jax.dev/en/latest/jax-101.html
https://pytorch.org/blog/overview-of-pytorch-autograd-engine/

Motivation

C++ still (mostly) the language of choice for performance-oriented scientific software

Long edit-compile-run cycles

Complex language syntax

Not interactive

autodiff

CoDiPack

Clad

Enzyme AD

boost C++ LIBRARIES

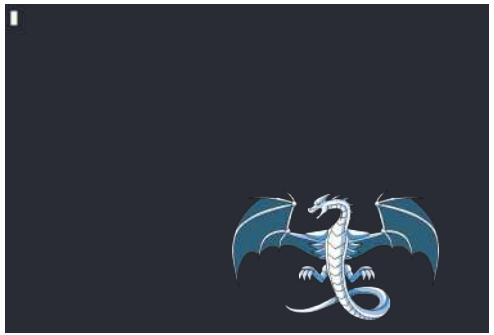
Motivation

C++ still (mostly) the language of choice for performance-oriented scientific software

~~Long edit-compile-run cycles~~

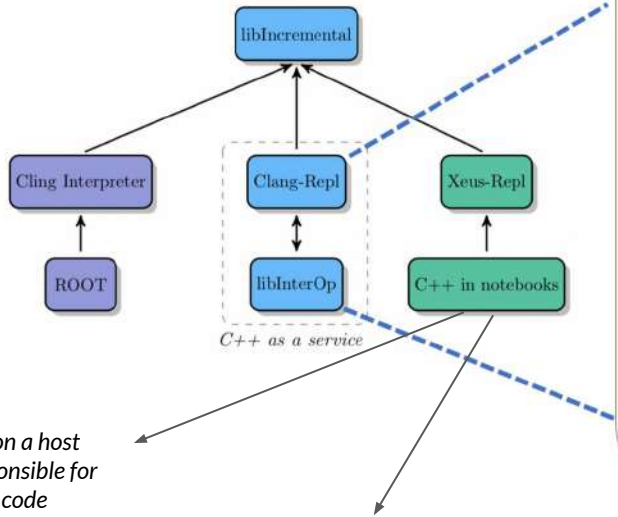
Complex language syntax

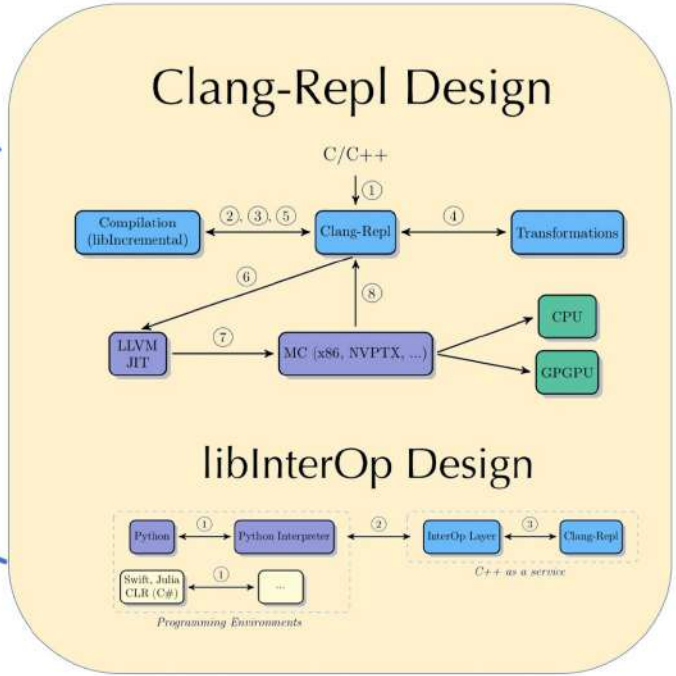
~~Not interactive~~



https://xeus-cpp.readthedocs.io/en/latest/
https://clang.llvm.org/docs/ClangRepl.html

Interactive C++?





Native

Deployment on a host machine responsible for executing the code

WebAssembly

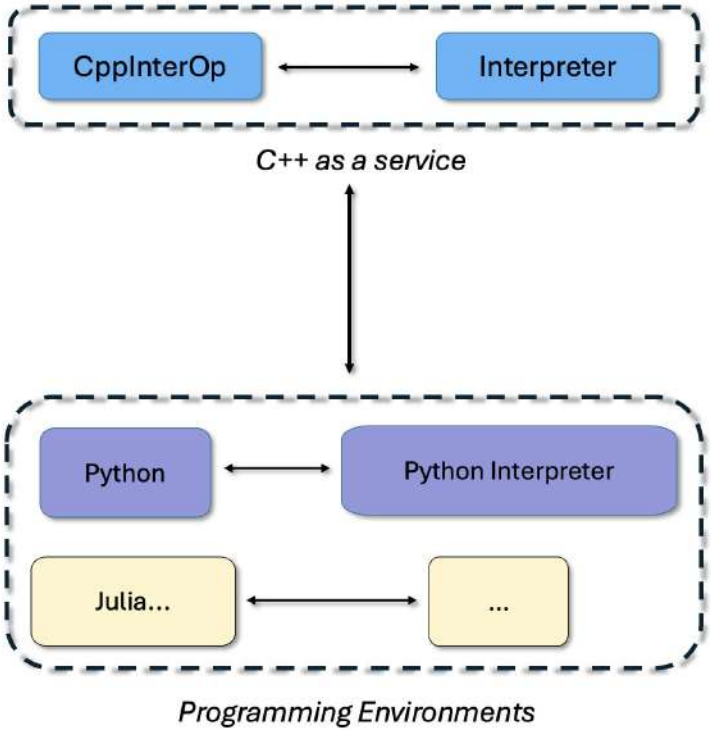
Hosted on a server, code executes on the browser. Low-level assembly-like language with a compact binary format, near native performance

CppInterOp

A lightweight layer on top of LLVM that provides efficient, on-demand reflection and a JIT compiler

Easily embeds Clang and LLVM as a libraries in your framework

Supports downstream tools that utilize interactive C++ by using the compiler as a service.



https://github.com/compiler-research/CppInterOp

Live Demo

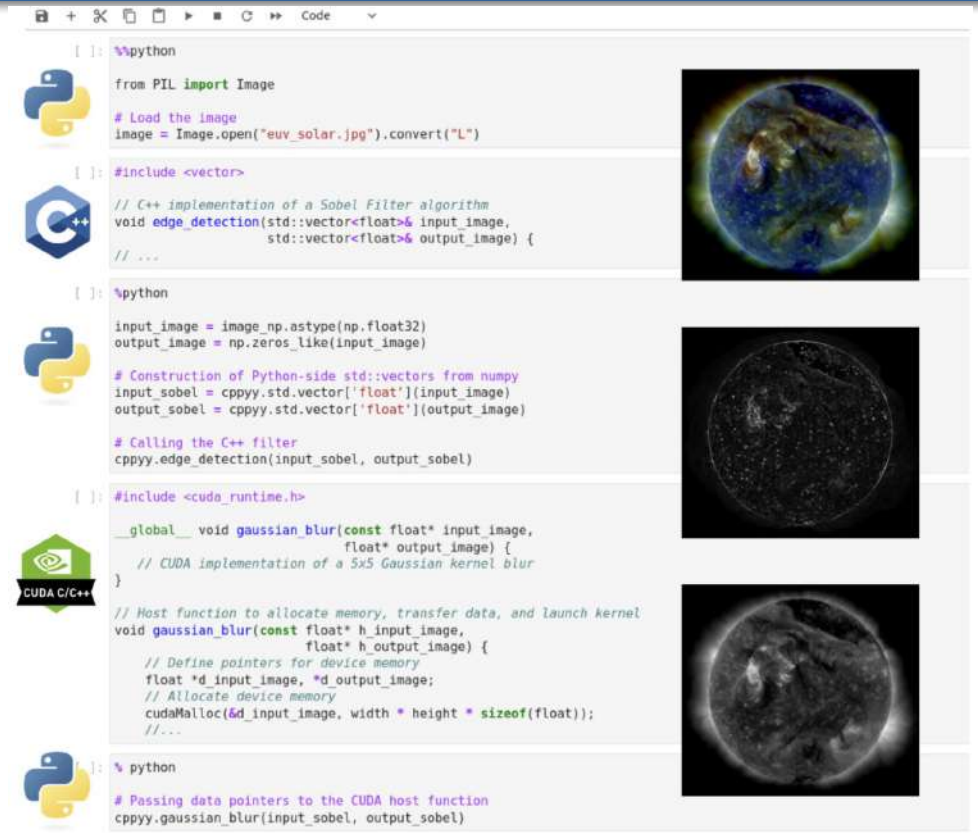
Rudimentary WebAssembly version (no included AD libraries) here: https://compiler-research.org/CppInterOp/lab/index.html

Please reach out for a reproducible docker container of the demo shown today

Future directions

- Integration of Python plugin based on *cppyy*
- Allows you to hop between the optimal language for each task
- Script host and device code on the fly
- Automatically bound to Python runtime

Possible, currently not in mainline



A. Jomy, B. Kundu et al, CppInterOp: Advancing Interactive C++ for High Energy Physics, https://doi.org/10.1051/epjconf/202533701165

Thank You!