Evolving ROOT's Python Interfaces

Interoperability with the Python ecosystem

Silia Taider[1], Vincenzo Eduardo Padulano[1]

[1] CERN EP-SFT

PyHEP.dev 2026 - 07-09-2026

https://root.cern

- **Python usage across the community is widespread**
- **ROOT is already a Python package – and aims to be a good ecosystem citizen**
- **Work on multiple fronts sets the direction:**
  - **A more Pythonic package**
  - **More interoperable with other Python tools**
  - More **easily obtainable** by Python users → contribution #31390 (tomorrow)
  - **Following** Python **standards** and best practices

In this talk

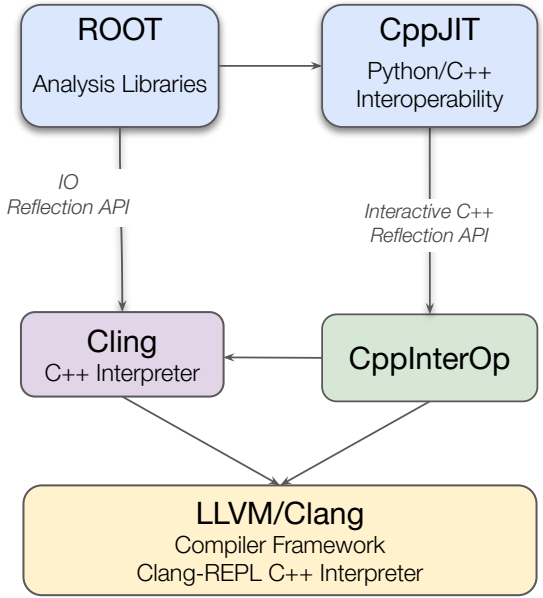
The bindings foundation, then interoperability across analysis, histograms and ML

Automatic & Dynamic Bindings

https://root.cern

Automatic and Dynamic Bindings

- **ROOT gives Python users automatic access to its C++ through runtime bindings**
- **CppInterOp: a novel Clang-based interoperability and reflection layer based on which the ROOT team and the Compiler-Research group modernized the cppyy bindings**
- **CppJIT: successor project to cppyy**
  - Provides automatic Python-C++ bindings
  - No domain-specific dependencies on ROOT and Cling
  - ROOT is moving towards this generalised technology as an external dependency
  - No string-parsing driven meta-type-system: opaque handles to C++ entities on the compiler AST
  - Fewer dependencies, more performant and crash-resilient
  - Enables a more complete feature set from the C++ standard

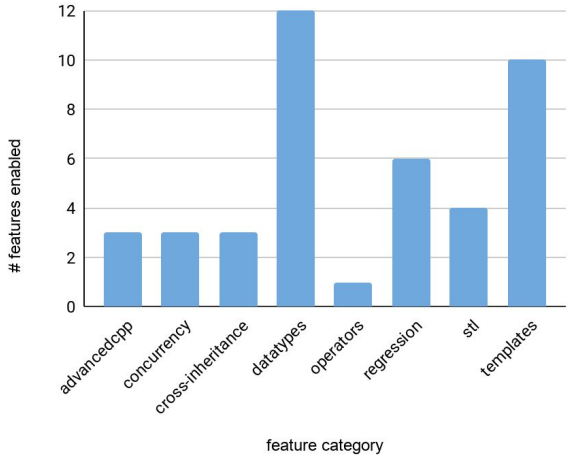


COMPILER C | R RESEARCH

For more info. see
ROOT's new Python Interfaces based on CppInterOp: Performance, Correctness, and Future-Proof Design
A. Jomy - CHEP 2026

What this changes

- **Conforms to LLVM/Clang**
  - Correct type conversion and overload resolution
- **More sustainable codebase, crash-resilience**
- **Unlocks more C++ idioms / usage patterns from Python**
  - Return smart pointer from callback
  - Derived classes access to protected members
  - Inheriting from a non-copyable base class
  - Use of class enum with typedef'd type
  - Templated binary operators
  - Use of ctypes for pass-by-ref/ptr
  - Support for complex numpy arrays
- **Fixes ~20% of open ROOT Python issues**
  - #19517 - Overloads like foo(MyEnum, MyClass) not correctly resolved
  - #19358 - cppyy.gbl.std.function fails for template types like ROOT::RVec<int>
  - #16852 - Regression in accessing protected data members in cross-inheritance
  - #10680 - Namespace dropped in std::function cppyy binding
  - #15062 - Non-trivial type traits do not work on cppyy
  - #15085 - PyROOT/cppyy fails with template functions overload with SFINAE
  - #15378 - Problem with templated binary operator implementations like operator+
  - #9246 - Enums are not correctly resolved when using an alias within a class/struct
  - #7077 - Incorrect/unstable overload resolution with enum class



ROOT is moving to a *new compiler-infrastructure* for more powerful Python bindings for it's developers and users

Pythonizations

https://root.cern

- **Pythonization:** a **customization layer** that expands automatically bound C++ classes with Pythonic behaviour (*such as context managers* 👇)

```
f = RFile.Open("myfile.root")
h = f.Get("my_histo")
f.Close()
```

💡 `f` is a dynamic **Python proxy** of a C++ object

Thanks to automatic bindings

```
With RFile.Open("myfile.root") as f:
    h = f["my_histo"]
```

💡 Access file content as a **Python dictionary**

Thanks to Pythonizations

Our Goal:

Enable fully Pythonic analysis With ROOT

-Today I show advancements in the analysis Pythonic API-

ROOT 6.38

- Use Numba to JIT-compile Python functions directly inside the RDataFrame event loop

```
@ROOT.Numba.Declare(["float", "float"], "float")
def get_mass(p, E):
  return np.sqrt(E**2 - p**2)

rdf = rdf.Define("m", "Numba::get_mass(p, E)")
```

💡 The decorator generates a wrapper

💡 Directly pass the wrapper to `Define` and `Filter`

```
@ROOT.Numba.Declare(["ROOT::Math::PtEtaPhiMVector"], "float")
def get_mass(v):
  return v.M()
```

💡 You can specify certain ROOT types

Level of Support for Python Callbacks

Legend:
✅ Supported
⚠️ Partial
❌ Not supported

What can the callback contain?	Support
Fundamental types	✅
C++ containers, NumPy and Awkward arrays	✅
ROOT Classes	⚠️
I/O inside the callback	❌
External Python libraries	❌
User-defined Python classes	❌

Supported as far as the ak-numba support goes

Limited by the cppyy numba_ext

Numba JIT-compiled Python functions into the C++ event loop *are fast but restricted*

EXPERIMENTAL

- **What if we can leverage the new Python bindings stack to run pure Python callables inside the RDataFrame C++ event loop?** 👉 CHEP 2026

  → New: pass a plain Python callable with type annotations → any valid Python runs

```
def get_mass(v: ROOT.Math.PtEtaPhiMVector) -> float:
  with open("log.txt", "a") as f:
     f.write(f"{v.E()}, {v.P()}\n")
  return v.M()

rdf.Define("mass", get_mass, ["v"])
```

💡 File I/O!

EXPERIMENTAL

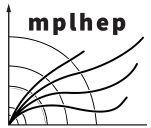
- **How far can we push RDataFrame's interoperability with HEP Python packages?**

  As far as your imagination takes you…

```
h = hist.Hist.new.Reg(...)

def predict(x: float, y: float, z: float) -> float:
    m = vector.obj(px=x, py=y, pz=z, E=2 * x).mass
    h.fill(m=m)
    mplhep.histplot(h)
    return float(jnp.tanh(jnp.float32(m)))

df = (
    ROOT.RDataFrame(...)
    .Define("score", predict, ["x", "y", "z"])
)
```



- **This feature is not available yet, but it's coming soon!**
  - Enabled by the `CppJIT` migration
  - Default: the Python interpreter

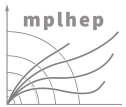
    → opt in to `@Numba` for the parts you want jitted

What can the callback contain?	Support
Fundamental types	✅
C++ containers, NumPy and Awkward arrays	✅
ROOT Classes	✅
I/O inside the callback	✅
External Python libraries	✅
User-defined Python classes	❌

ROOT Implements the UHI

ROOT 6.36

- The Unified Histogram Interface (UHI) is a specification to unify histogram operations across Python libraries
- **ROOT adopted UHI in v6.36**
  - Implements indexing and plotting
  - Thanks to H. Schreiner for engaging with us



```
# slice the first 10 bins of X
h3[:10, ...]
# slice and rebin X, project Y
h3[10:20:rebin(2), ::sum, ::]

# h2 is ROOT.TH2F
sns.heatmap(h2.values(), ...)
# h1 is a ROOT.TH1D
hep.histplot(h1, ...)
...
```

UHI Serialization

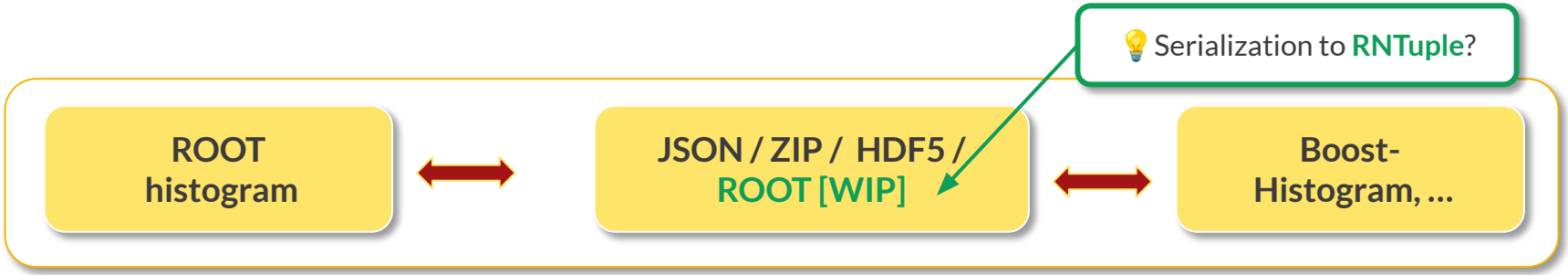
- UHI enables **exchanging ROOT histograms across Python libraries via a shared format**

```
h_root = ROOT.TH1D("h", "h", 10, -5, 5)
... # fill with some data

ob = json.dumps(h_root, default=uhi.io.json.default)
ir = json.loads(ob, object_hook=uhi.io.json.object_hook)

h_loaded = hist.Hist(ir)
```

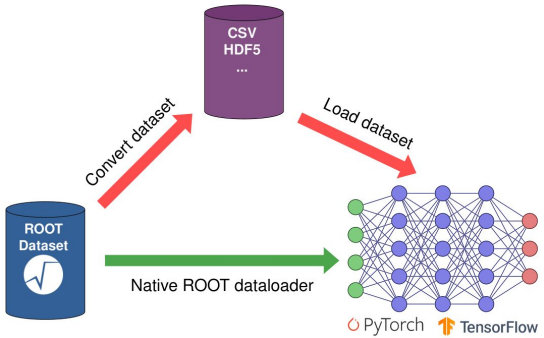
💡 Create a Hist histogram out of a ROOT histogram



- **A native data loader for ROOT**: `ROOT.Experimental.ML.RDataLoader`
  - TTree & RNTuple → Training batches directly, no conversions
  - Outputs: NumPy arrays, PyTorch tensors, TensorFlow datasets, JAX arrays
  - 👉 Available today 👉 CHEP 2026

- **Validated across LHC experiments and future-detector R&D:**
  - *ATLAS - EgammaCNN trigger classifier (NextGen Triggers, L0Global HL-LHC)*
    - **4.8x faster training**, equivalent physics performance
  - *FCC-ee - ALLEGRO calorimeter fast simulation (CaloDiT-2)*
    - **~100% GPU utilization** sustained; no more conversion overhead

  Across other use-cases: no format conversion, no custom loading code, faster and more efficient training depending on the configuration



Train ML models from ROOT data very efficiently
Being developed and validated with experiments

ROOT 6.40

```
dl = RDataLoader(
    rdf,
    batch_size=128,
    batches_in_memory=100,
    target="label",
)

train, test = dl.train_test_split(test_size=0.2)

for X, y in train.as_torch(device="cuda"):
    model(X, y)
```

Shuffle & memory knob
Higher = better mixing
Lower = less RAM

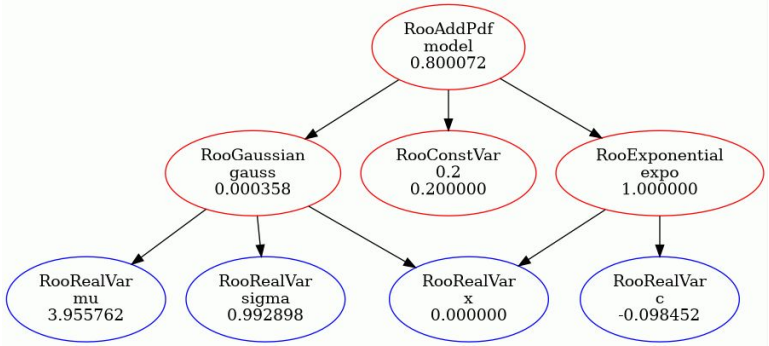
4 output formats
Also: `as_numpy()`
Also: `as_tensorflow()`
Also: `as_jax()` ROOT 6.42

Statistical inference with ROOT

RooFit: C++ library for statistical data analysis in ROOT

Tailored to the HEP community needs:

- Likelihood functions highly optimized for the context of minimization with Minuit
- It takes care of analytical normalization integrals where possible
- User-extensible framework that can cover a wide range of use cases
  - *Binned* likelihood fits
  - *Unbinned* likelihood fits
- Sharing of statistical workspaces thanks to ROOTs powerful IO system
- All visible and usable from Python!



Conditional pdf example:

$$p(x|y) = \frac{p(x,y)}{p(y)} = \frac{p(x,y)}{\int p(x,y)dx}$$

Observable subdomain example:

$$p(x|\text{subrange}) = p(x)\frac{\int_{\text{full}} p(x)dx}{\int_{\text{subrange}} p(x)dx}$$

- RooFit integrates with other Python tools and moves along the latest research in the community
- New with ROOT 6.40! RooONNXFunc wraps any ONNX model in RooFit
  Immediately applicable to Neural Simulation Based Inference (NSBI)

```python
import ROOT

# Define variables
x = ROOT.RooRealVar("x", "x", 0.0)
y = ROOT.RooRealVar("y", "y", 0.0)
z = ROOT.RooRealVar("z", "z", 0.0)

# Create ONNX function
func = ROOT.RooONNXFunc(
    "func", "func",
    [[x, y], [z]],
    "model.onnx"
)

# Evaluate
print("Model output:", func.getVal())
```

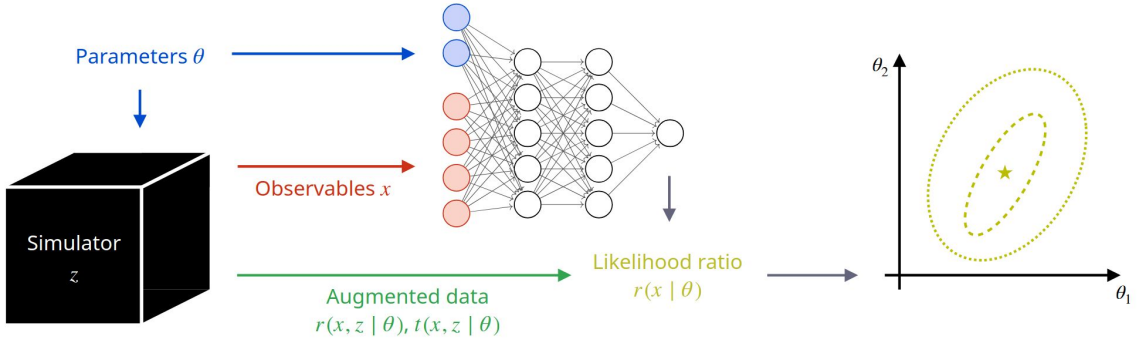


Figure borrowed from Alexander Held's talk at the PHYSTAT-SBI 2024 workshop

Conclusion

- **ROOT is becoming a more pythonic package than ever before**

  → aiming at seamless integration with the scientific Python ecosystem
- **The `CppJIT` migration enables more powerful pythonizations and new idioms**
- **This applies to different components of ROOT: analysis interfaces, histogramming, machine learning…**

How can we make ROOT even more interoperable?

Thank you!

https://root.cern