

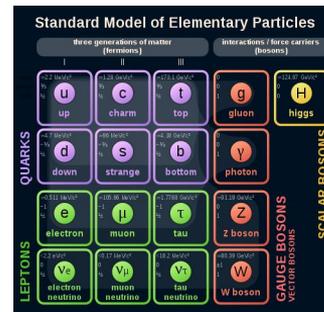
How Automatic Differentiation can help Particle Physics

*Jonas Rembser*¹

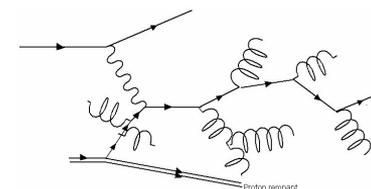


Particle physics analysis an a nutshell

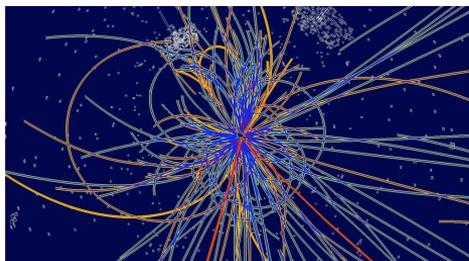
- ▶ Developing physics **theories** that make predictions
- ▶ **Simulating** collision data and detector responses from models
- ▶ Experiments **taking data** and reconstruct events
- ▶ **Analysis** comparing data with simulations to refine theory and guide future measurements



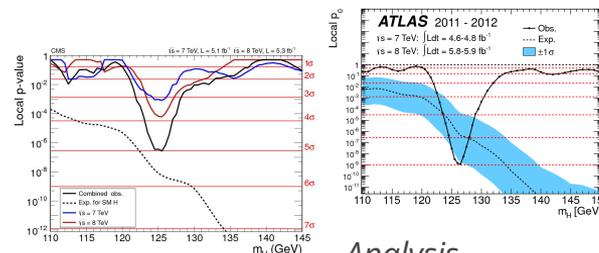
Theory



Simulation



Data taking



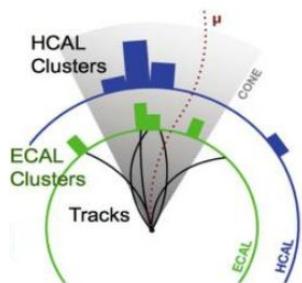
Analysis



Simulation-driven analysis

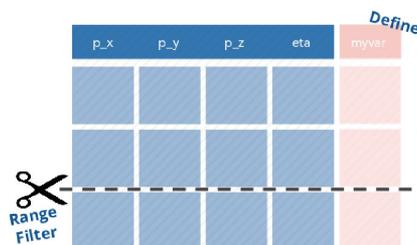
Event data is used in **simulation-driven** data analysis pipelines

- ▶ Data reduction usually involves **histogramming**
- ▶ Surrogate models traditionally built as *interpolations* of histogram stacks
 - ML surrogate models on full event data are also becoming popular



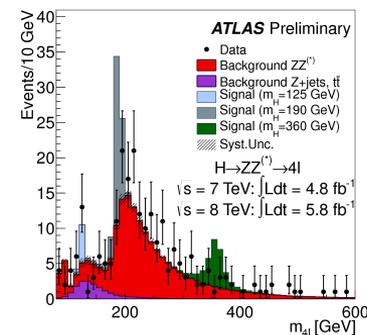
Event simulation and reconstruction

Usually managed centrally by experiments computing.



Event data reduction for analysis need ("event loop")

Done by data analysis groups, distributed on grid.



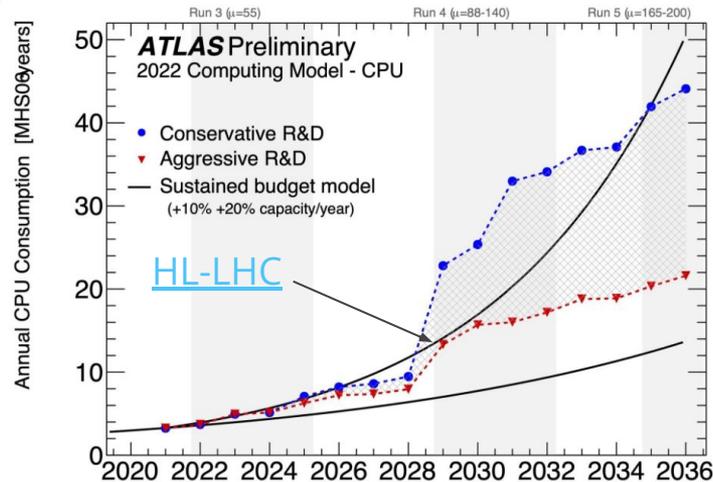
Statistical Inference

Physicists often do what they want, but have to convince collaborators of validity.

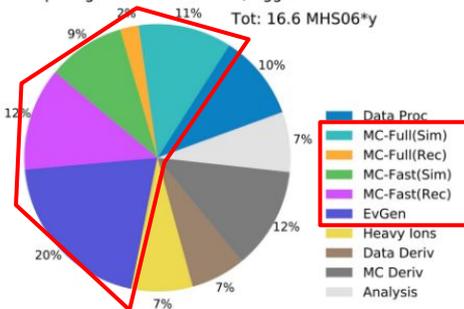


AD Motivation - Future Collider Challenges

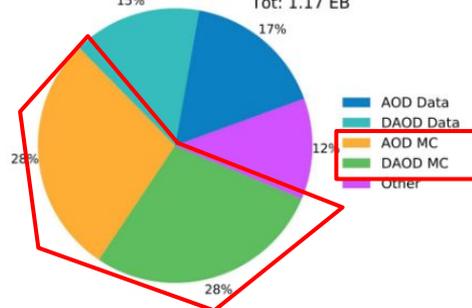
- ▶ **Experiments scaling up:** HL-LHC and future colliders - orders-of-magnitude more data & complexity. Computing needs far outpace flat resource budgets.
- ▶ **Current software stack:** Decades-old HEP frameworks (simulation, reconstruction, analysis) reaching performance limits. New approaches needed to close the resource gap.
- ▶ **Opportunity:** Leverage computing tools like automatic differentiation to maximize efficiency and physics sensitivity.



ATLAS Preliminary
2022 Computing Model - CPU: 2031, Aggressive R&D
Tot: 16.6 MHS06*y



ATLAS Preliminary
2022 Computing Model - Disk: 2031, Aggressive R&D
Tot: 1.17 EB

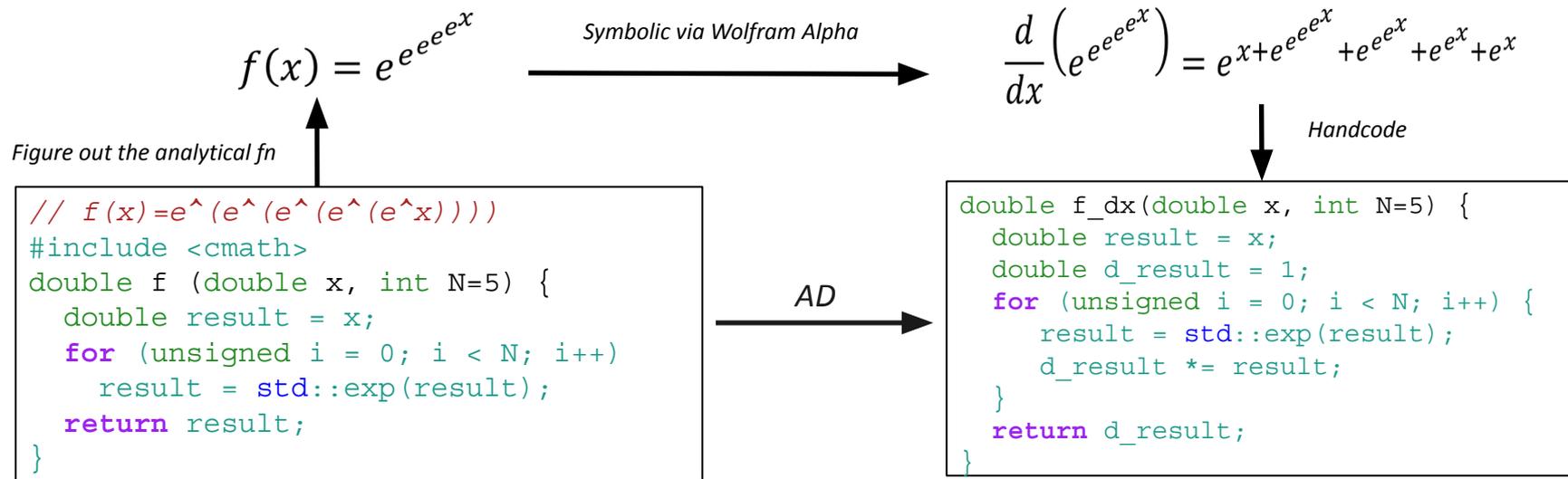


From [ATLAS computing roadmap](#)



What is Automatic Differentiation (AD)?

- ▶ **AD Basics:** Technique to compute exact derivatives of complex code efficiently
- ▶ **Gradient-Based Optimization:** Enabled by AD, widely used to train neural networks
- ▶ **Differentiable Programming:** Treat the entire analysis or simulation as a differentiable program, enabling end-to-end gradient-based optimization of parameters

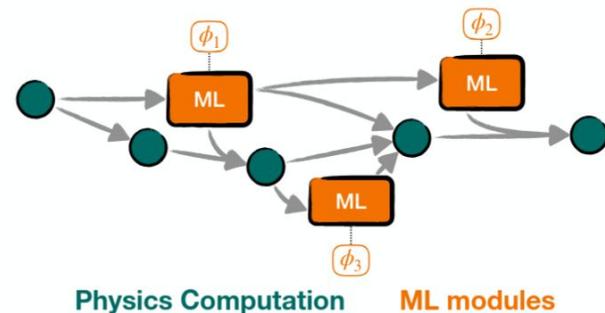


Reference: V. Vassilev - Accelerating Large Scientific Workflows Using Source Transformation Automatic Differentiation



Why AD in HEP? Potential benefits

- ▶ **Faster & More Stable Fitting:** Exact gradients can dramatically speed up likelihood maximization and improve convergence stability.
- ▶ **Optimizing Detector and Algorithm Design:** e.g.:
 - tweak detector geometry for better resolution
 - tune reconstruction algorithms by optimizing a differentiable performance metric
- ▶ **Global optimization** of physics computation and ML modules in the processing pipeline
- ▶ **Get the most out of simulation** (*aka. better labels*): Additional derivatives in simulation can for example:
 - Enhance efficiency of training *surrogate models* for statistical analysis
 - Be used to *extrapolate to different parameter states* without re-simulating
- ▶ **Optimize with respect to final physics goals**, considering also systematic uncertainties (*aka. better losses*)
 - E.g., end-to-end **differentiable analysis**, including the statistical models for systematic uncertainties to **optimize** algorithm parameter **for discovery significance**



For this to work, the physics components must **play nice gradient descent** → differentiable programming

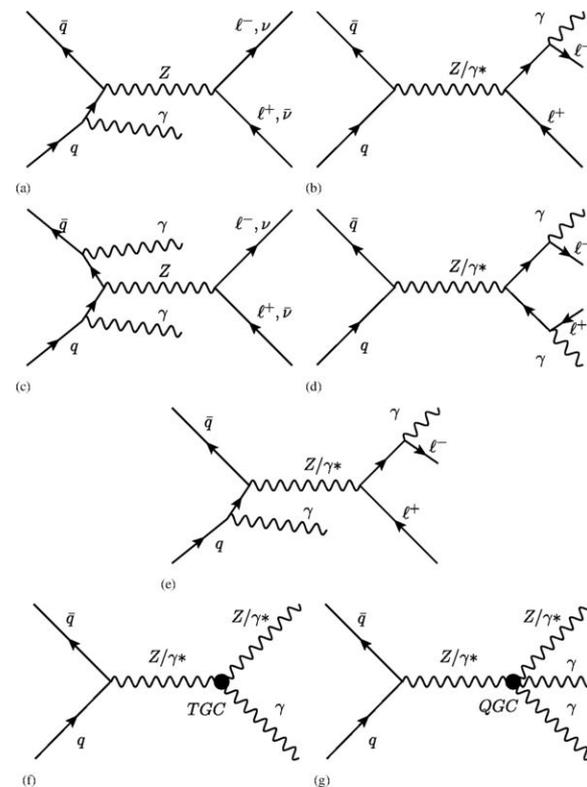
Figure borrowed from L. Heinrichs plenary talk at ACAT 2024



Physics Process Generation

Several software tools to compute the **hard scattering**, e.g.:

- ▶ Software mostly developed by *theorists*
- ▶ Many tools, like MadGraph5_aMC@NLO, POWHEG Box, WHIZARD, etc.
 - Specialized for different processes
 - Some more general than others
 - Experiments often use multiple tools for cross-validation
- ▶ Some tools like MadGraph use Fortran/C++ code generation for matrix element code



Illustrative Feynman Diagrams from ATLAS $Z\gamma\gamma$ paper:
[10.1103/PhysRevD.93.112002](https://doi.org/10.1103/PhysRevD.93.112002)

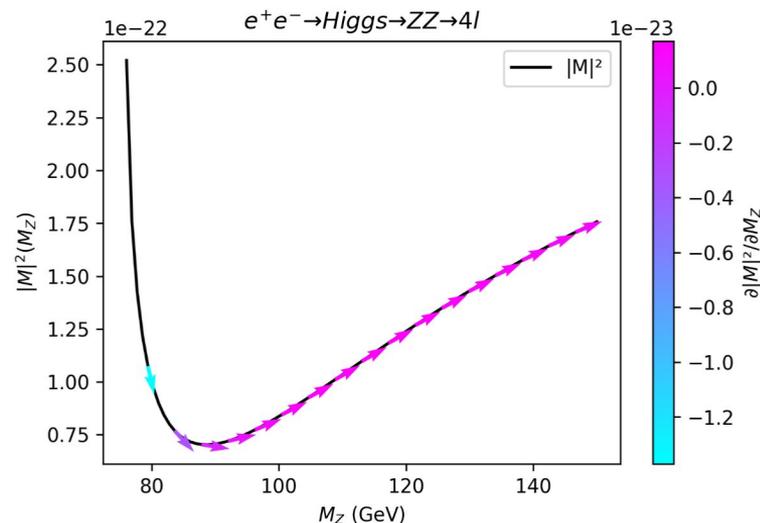


MadJax: a prototype to success

- ▶ **MadJax:** instead of producing Fortran or C++ code for matrix element calculation, produce Python code that uses JAX via custom MadGraph plugin
- ▶ **Performant:** using the JAX JIT, the resulting code is close to C++ in performance
- ▶ **Limitations:** the reliance on JAX creates a language barrier that can be a problem for downstream simulation code (*like Geant4*)

Citing from the MadJax paper (2022):

“JAX is not always well-adapted to the structure of matrix element code. Future directions will explore the use of alternative AD tools that may be better suited to matrix element code structures.”

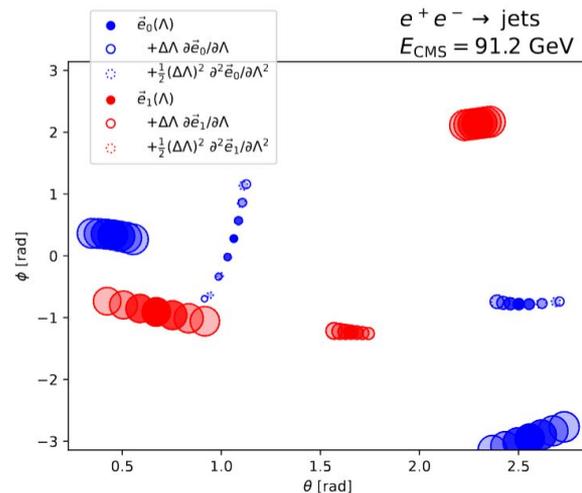


Example of derivative with respect to physics parameter, from [differentiable matrix elements with MadJax](#)



Parton showering and Hadronization

- ▶ Parton showering and hadronization: **stochastic** simulations with **many hyperparameters** that encode systematic uncertainties
- ▶ Different algorithms available
 - Diffability can be achieved with smart algorithm design that **factorizes randomness and parameter dependence** (like the [EventMover](#))
- ▶ Variations w.r.t. hyperparameters often encoded by parametric models for downstream analysis
 - Bypassing need for gradients at cost of approximation
- ▶ Differentiation w.r.t. hyperparameters can be interesting for **automated generator tuning** (factorization scale, Pythia tunes, etc.) or **systematics parametrization** with Taylor series



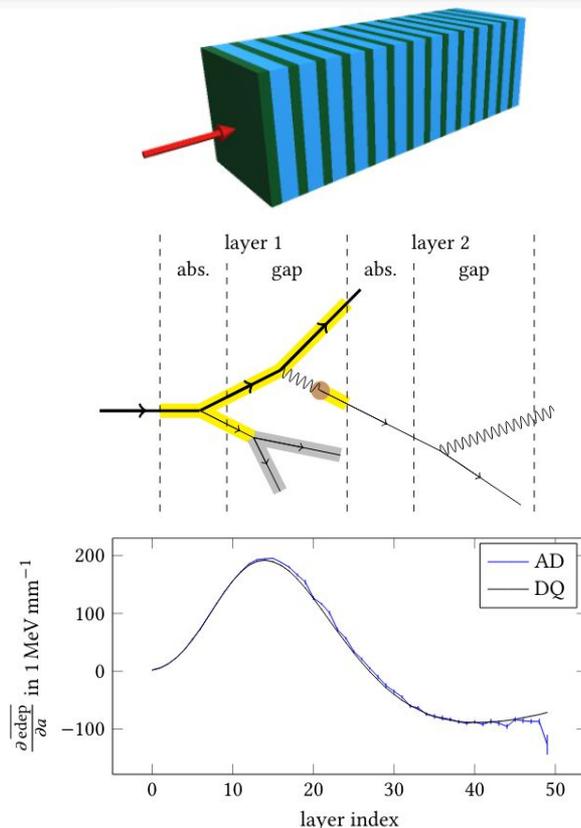
From [arXiv:2208.02274v1](#) "EventMover"



Detector Simulation

Detector simulation is usually done:

- ▶ With the **Geant 4 simulation toolkit**, simulating all particles interaction with the detector material (*aka. "full sim"*)
- ▶ With parametric models, tailored to experiments of in general tools like **Delphes** ("*fast sim*")
- ▶ Differentiating full simulation is challenging because of **stochasticity**
 - Motivating research on AD of programs with randomness (e.g. [arXiv.2308.16680](https://arxiv.org/abs/2308.16680))
 - Recent publication [arXiv.2405.07944](https://arxiv.org/abs/2405.07944) showed **proof-of-concept for differentiation of electromagnetic shower** simulation w.r.t. *geometry and initial particle parameters*

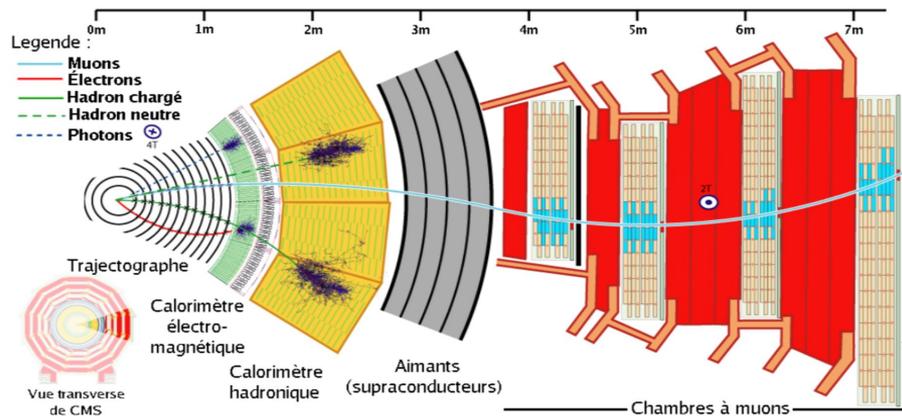


Figures from [arXiv.2405.07944](https://arxiv.org/abs/2405.07944)

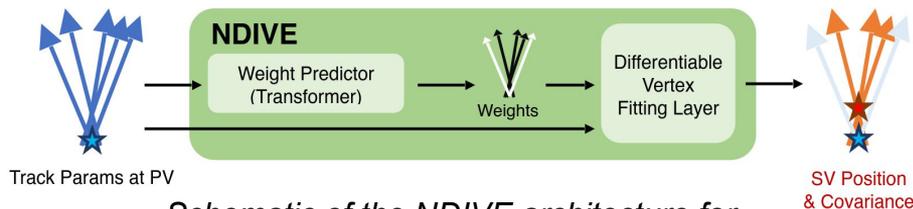


Event Reconstruction

- ▶ Event reconstruction combines multiple algorithms, roughly:
 - **Track** and **vertex** reconstruction
 - **Calorimeter** clustering
 - **Matching** both together and classify particles
- ▶ Differentiable algorithms help to **holistically optimize ML components** in the reconstruction chain
 - See e.g. [differentiable vertex fitting for flavor tagging \(NDIVE\)](#)
- ▶ Having **common reconstruction software** between experiments can help to *free up efforts* for implementing gradients
 - For example, in the [ACTS](#) common tracking software



Overview on event reconstruction with the CMS detector.



Schematic of the NDIVE architecture for vertex fitting.



Statistical inference based around the **likelihood**.

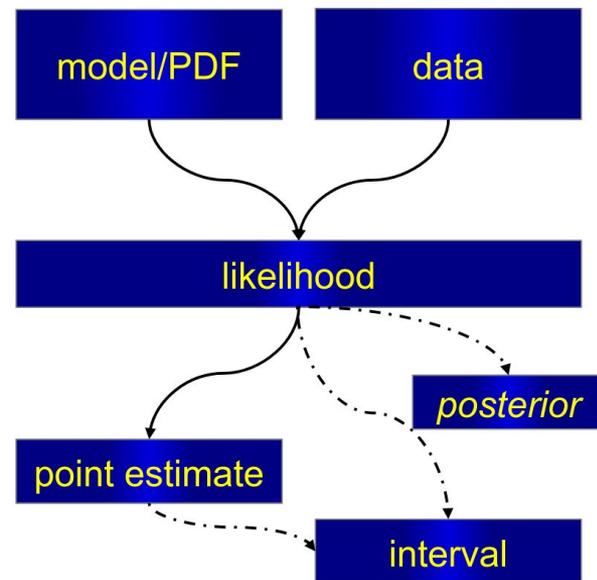
Different **frameworks** for various kinds of fits, e.g.:

- ▶ [RooFit](#): C++, general purpose, AD with Clad
- ▶ [pyhf](#): Python, binned likelihood fits, AD with JAX
- ▶ [zfit](#): Python, mostly unbinned fits, AD with TensorFlow

Likelihood generally contains **unbinned components**, **binned components**, and **constraints** from parameterized auxiliary measurements.

$$L(\vec{n}, \vec{a} | \vec{\eta}, \vec{\chi}) = \prod_{c \in \text{unbinned}} \prod_{i \in \text{Obs}} \frac{f_c(\vec{x}_{ci} | \vec{\eta}, \vec{\chi})}{\int f_c(\vec{x}_{ci} | \vec{\eta}, \vec{\chi}) d\vec{x}_c} \cdot \prod_{c \in \text{binned}} \prod_{h(\text{analytical})} \prod_{b \in \text{Obs}} \text{Pois}(n_{cb} | v(\vec{\eta}, \vec{\chi})) \cdot \prod_{\chi \in \vec{\chi}} c_\chi(a_\chi | \chi)$$

\vec{n} : data, \vec{a} : auxiliary data, $\vec{\eta}$: unconstrained parameters, $\vec{\chi}$: constrained parameters



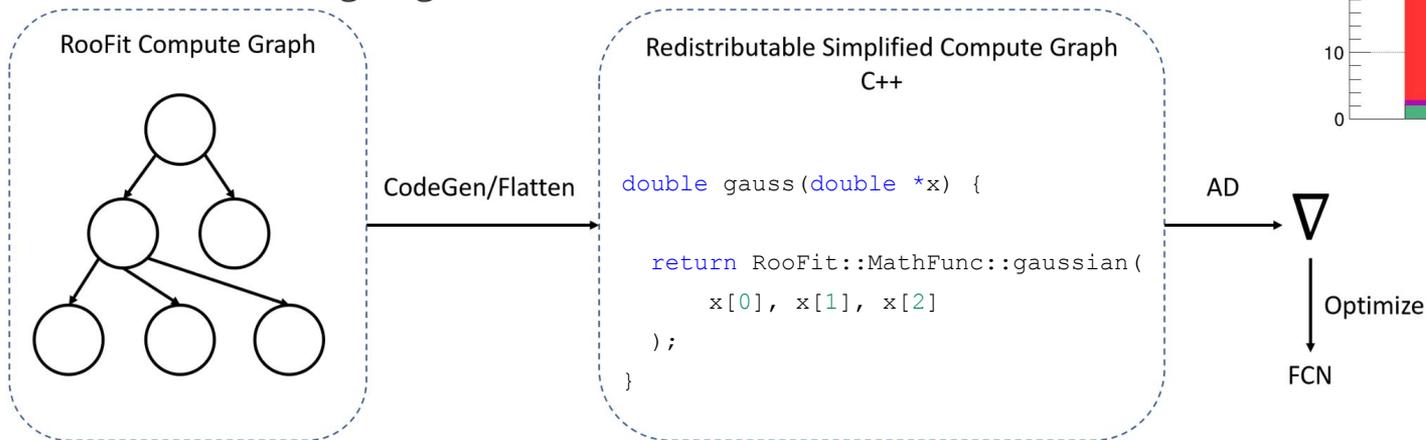
Flowchart borrowed from [talk by J. Eschle](#)



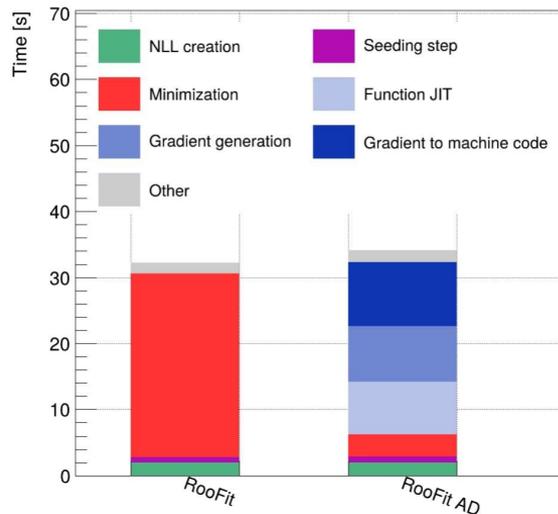
AD in RooFit with Clad

RooFit uses [Clad](#) to get analytic gradients for likelihoods: **Code generation** (see [ROOT March 2025 blog post](#))

- ▶ Shows we *can* integrate AD in **existing C++ frameworks** with no changes required by users!
 - *Transparency* is key for adoption
 - The engineering cost is still not negligible
- ▶ **Up to 10x speedup** in binned likelihood fit benchmarks
- ▶ **No numerical tricks required** for convergence anymore, like offsetting large numbers and Kahan summation



ATLAS HistFactory fit *ROOT 6.38.00*



Minimization time for ATLAS Higgs combination benchmark (49 HistFactory channels, 739 parameters in total, in [rootbench](#)) with and without AD.

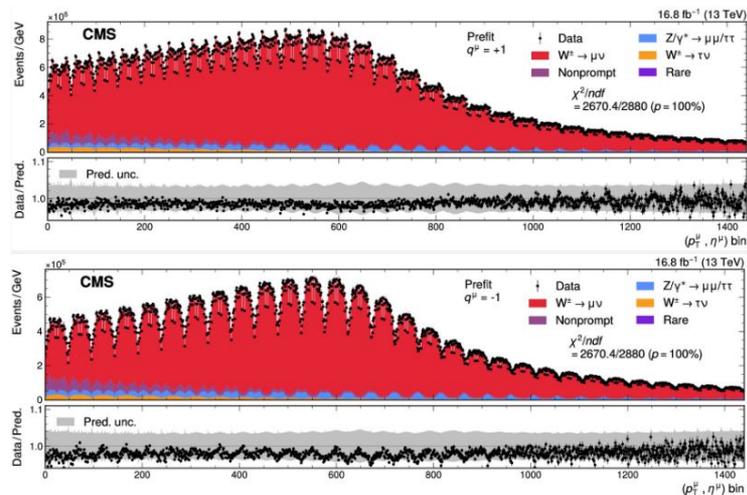
From the presentation on AD in RooFit on the [2025 ROOT users workshop](#).



Outlook on statistical inference

Statistical analysis tools need to evolve to accommodate for ever-increasing fit complexity.

- ▶ Physicists are thinking again about different ways to do likelihood minimization
- ▶ One particular example is the **recent W-mass measurement by CMS** (see [talk at the ROOT workshop](#) for technical details)
 - Uses *TensorFlow* for analytic **Hessian-vector products**, with **Trust-Krylov minimizer** from SciPy
 - Quite different from the usual RooFit+Minuit 2
- ▶ The **inverse Hessian** of the likelihood is used estimating parameter uncertainties, but evaluated only once
 - Not performance critical, but numeric Hessians a challenge if *eigenvalues of very different order of magnitude*
 - => might have to rely on **AD for accurate results**

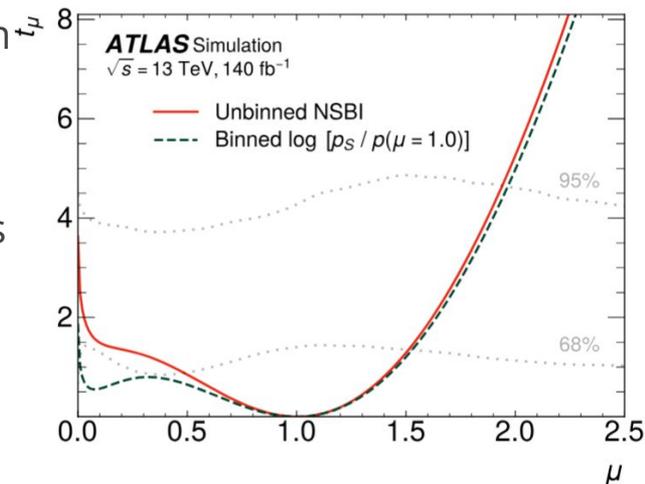


Example of a complex binned likelihood fit: the W-mass measurement at CMS (from the [PyHEP 2025 presentation on rabbit](#))



Simulation-based inference

- ▶ **Simulation based inference (SBI)**: inference procedure that relies on sampling from a generative model
 - Particle physics *has done this for decades*, using **morphed template histograms** filled from simulation
- ▶ Using **neural network likelihood surrogates** instead of histograms extends the feasible analysis problem space (**neural SBI**)
- ▶ The usage of AD **is for the neural network**, *not* the physics model or statistical analysis
- ▶ **SBI is not directly “AD for particle physics”**. It’s “AD for deep learning for particle physics”, just like jet tagging.
- ▶ The **synergy with AD** lies in augmenting simulation data with gradient info for better neural SBI (*the [MadMiner paper](#) is a good reference*).

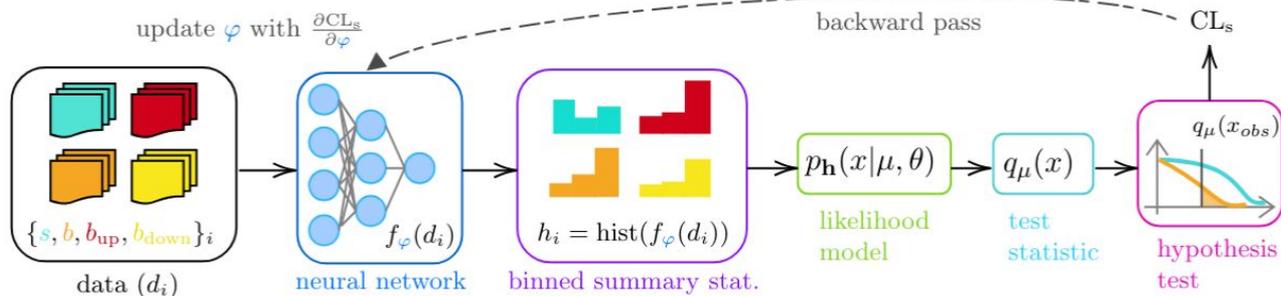


[ATLAS has shown](#) how neural SBI improves over template histogram fits for their off-shell Higgs Boson coupling measurements.



Analysis optimization

- ▶ **Assume** the test statistic in the **statistical inference is differentiable**
- ▶ Event data reduction usually involves tunable parameters
 - e.g. neural networks, but could also be simple selection cuts
- ▶ We can build an **inference-aware loss** and **optimize the analysis** to get the most sensitive hypothesis test
 - Demonstrated for simple examples in [j.cpc.2019.06.007](https://arxiv.org/abs/1906.007) and [arXiv:2203.05570](https://arxiv.org/abs/2203.05570)



From [arXiv.2203.05570](https://arxiv.org/abs/2203.05570).



Detector Optimization

“While automated optimization systems cannot at the present time have the ambition to substitute the hand and the intuition of the expert, they cannot any longer be ignored as assistants to the design task.”

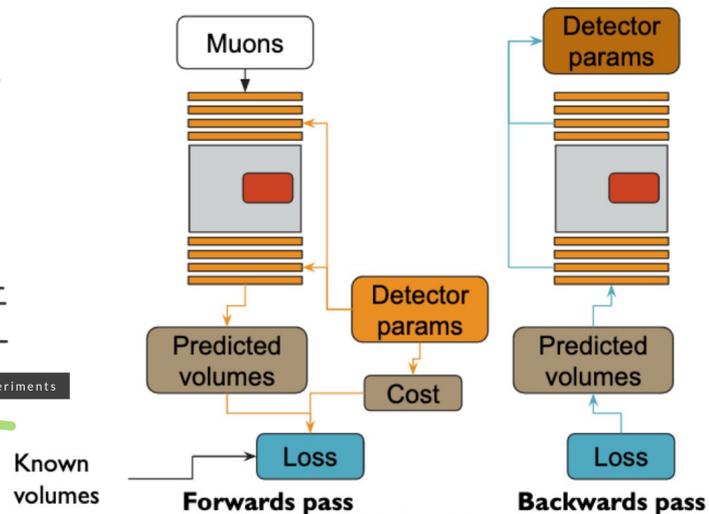
From the [MODE collaboration White Paper](#)



Assume:

- ▶ Simulation can be **differentiated w.r.t. the detector geometry** parameters
- ▶ Physics goal can be encoded in a differentiable **loss function**

We get automated detector design!



Excellent R&D by the [MODE collaboration](#), for example on geometry optimization for muon tomography ([TomOpt](#)).

See also [overview talk at ICHEP 2024](#) by P. Vischia.



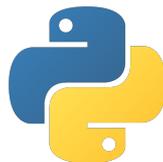
Target programming languages

On *which programming languages* to focus?

It is at this time **unrealistic to assume** that the software stack of a particle collider experiment can be built **without C++ in the data simulation and reconstruction chain**.

- ▶ **Julia** has limitations due to JIT and garbage collection, ecosystem still maturing
- ▶ Performance in **Python** often requires libraries specialized for ML/AI and *not* for HEP

But languages with a strong AD ecosystem like **Python** and **Julia** are **great for analysis!**





AD Tools and Frameworks

- ▶ **C/C++ AD Options:** Operator overloading libraries (e.g. `AdePT`, `ADOL-C`) or source transformation tools like `Clad` (Clang plugin), `Enzyme` (LLVM plugin), or `Tapenade`
 - Can be injected into existing C++ algorithms, but may struggle with data structures and control flow in big workflows
- ▶ **Python ML Frameworks:** Many R&D efforts re-implemented physics computation in Python to use autograd (TensorFlow, PyTorch, JAX)
- ▶ **Julia and others:** New languages like Julia offer solid built-in AD support. Most adoption so far by the theory community.

Clad

PyTorch



TensorFlow

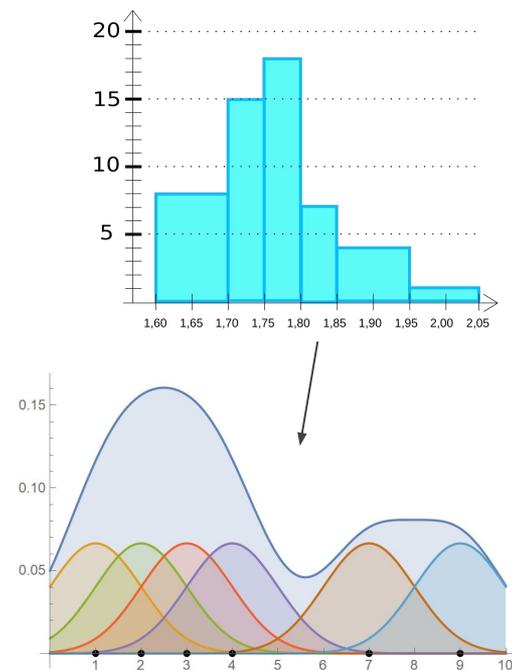
Final strategy likely a *combination*, picking the right tool for each job:

- ▶ Retrofitting AD into critical C++ components (e.g. via `Clad/Enzyme`)
- ▶ Rewriting some analysis parts in Python/Julia



Challenges towards differentiable HEP

- ▶ **Discontinuous Physics:** Many non-differentiable pieces (e.g. choices like “number of jets”). AD often requires smooth approximations (*starts already with histogramming!*)
- ▶ **Stochasticity:** Randomness (Monte Carlo sampling) yields high-variance gradient estimates. Ensuring unbiased gradients in a Monte Carlo remains a frontier topic.
- ▶ **Changes of analysis strategy required:** to make best use of gradients, new analysis techniques like differentiable (“relaxed”) approximations or neural SBI are required
- ▶ **Code patterns, performance and memory:** Naively differentiating a full experiment frameworks doesn't work. Codes were *not* written with differentiability in mind, and AD tools are often not optimized for the HEP use case.
- ▶ **Integration Effort:** HEP software is complex with long support windows (decades!) – adding AD must not destabilize it. Also, many pieces must work together for holistic optimizations.



Already going from histograms to KDEs would not be uncontroversial:

- *no Poisson statistics*
- *requires new hyperparameter*



One particular challenge for *differentiable simulation* for physics analysis:

- ▶ We are **victims of our own success**: decades of generator development have produced powerful reweighting machinery that mimics differentiability, so the real need is often hidden.
- ▶ In other words: it's *challenging* to find cases where augmenting simulation with AD gradients can save simulation cost, because:
 - Event **weights are already parameterized** (approximately or discretely) in many dimensions, usually nuisance parameters
 - Many analyses rely on *standard tune variations* or envelope uncertainties rather than continuous parameter fits
 - Weight dependence on physics parameters is often *analytically simple* (e.g. EFT: quadratic; ME: analytic reweighting).

To make a strong case for full diff'ability, **we need concrete analysis scenarios** where current reweighting breaks down, and those scenarios are rare or subtle.



From Prototypes to Production

My definition of *in production*:

Automatic differentiation (AD) is used in production if it is part of the validated analysis workflow of a collaboration-approved particle physics result.

- ▶ It's clear that AD is already widely used in production in particle physics **indirectly through general deep learning**:
 - From surrogate models on simulation, over ML modules in reconstruction, to simulation based inference and beyond
- ▶ Usage beyond deep learning mostly around:
 - **Derivatives of likelihoods** for fast and accurate statistical analysis

We have some **excellent prototypes** to address domain-specific problems with AD! 🙌
Which ones are worth to push towards production use, e.g. for HL-LHC?



- ▶ Injecting **automatic derivatives** can **improve performance** and enable **new analysis techniques** for Particle Physics
 - Some **examples**: Automated design, augmenting simulation data, faster likelihood minimizations, analysis-informed loss functions, combining neural nets with physics code, etc.
- ▶ Yet, most efforts are **prototypes**, especially in the simulation area
 - it's **time to push these prototypes to real use!**
- ▶ **The the AD experts**: help us adapt your tools to our complex use-cases
- ▶ **To the HEP developers**: be open to modernizing legacy code and workflows with gradient capabilities
- ▶ Keep in mind that the field spent *decades optimizing analysis pipelines* to circumvent the need for AD!

For the next HL-LHC data runs, let's aim for some *gradients flowing through the experiments simulation and reconstruction pipelines!*