Clad applied to likelihood fits with Minuit 2, RooFit, and CMS Combine

Jonas Rembser

CERN, EP-SFT



CompilerResearchCon 2026
4 November 2026

EP-SFT
Software Frameworks and Tools

This talk

Every LHC measurement ends in a **likelihood fit**, and the fit is driven by **derivatives**: gradients at every step, second derivatives for the uncertainties.

Three places where Clad meets the fit, then a wishlist:

1. **Minuit 2**, the minimizer: what it asks of the likelihood, and where it still falls back to numerical differentiation
2. **RooFit**: Clad inside the code generation, the benchmarks, and the milestone of the year, **analytical Hessians** with Clad 2.5
3. **CMS Combine**: a full AD pipeline for most CMS analyses, built in hackathons
4. **Future directions**: what the next steps need from Clad – Hessian-vector products, GPU kernels

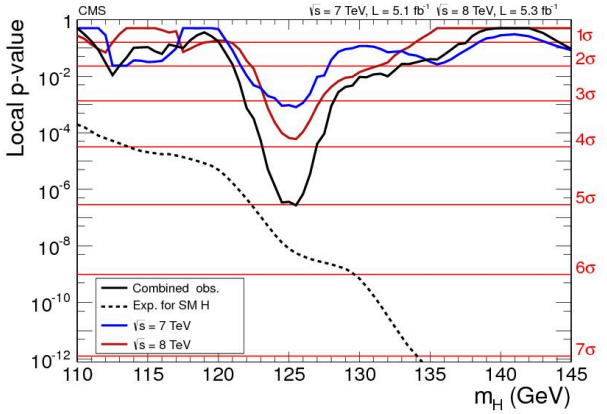
Last presented at a Compiler Research meeting in June 2025, the gradient story: this is what happened since, and where to join in.

RooFit and the likelihood fit

RooFit, the statistical modelling and fitting library of ROOT: nearly every ATLAS and CMS result is a likelihood fit built with it, the CMS ones through **Combine** (part 3).

One object: the **likelihood** of the observed data given the model parameters, $L(\theta) = \prod_i p(x_i \mid \theta)$

- **Fitting** = minimizing $-\log L$ over **all** parameters: a few of interest, hundreds to thousands of **nuisance parameters** – the systematic uncertainties, fitted but not reported
- **Uncertainties** from the curvature at the minimum (Hesse) or from **profile likelihood** scans: re-minimizing over everything else at each fixed value of one parameter
- Scans, toy studies (fits to simulated data), impacts (one refit per nuisance parameter): **one likelihood, many minimizations**



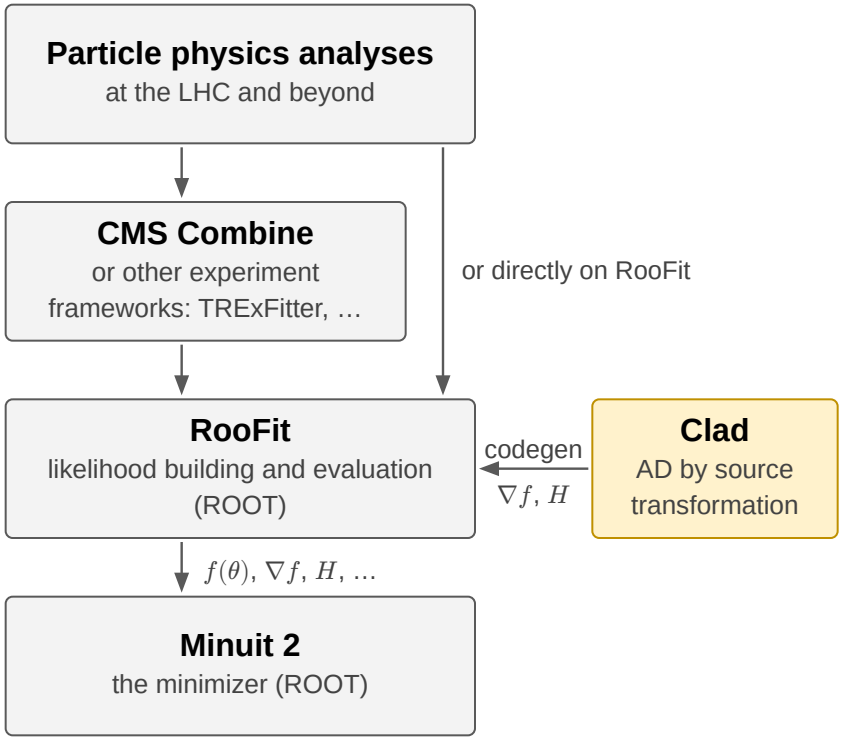
CMS, July 2012: the local p-value against the Higgs boson mass, computed by Combine on RooFit likelihoods.

RooFit was used in the Higgs boson discovery!

Where Clad enters the stack

- **RooFit** builds and evaluates the likelihood, and hands Minuit 2 the function, its gradient and now its Hessian
- **Clad** differentiates the C++ that RooFit generates for a likelihood

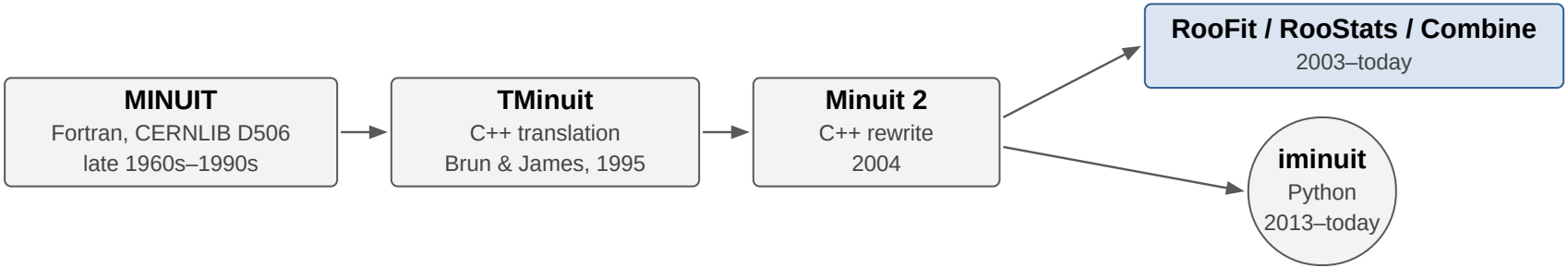
A derivative Clad produces travels down this stack; one that nobody provides is taken by finite differences.



Part 1 – Minuit 2: what the minimizer needs

Minuit 2, the minimizer of particle physics

- **Minuit** (Fred James, CERN): the last step of most HEP measurements for fifty years; **Minuit 2** is still the default under RooFit and Combine
- One interface: a function $f(\theta)$, here $-\log L$, and optionally its gradient – since ROOT 6.28 also its Hessian
- **Migrad** finds the minimum, **Hesse** measures the curvature there, **Minos** profiles for asymmetric errors
- Every derivative the function does not provide is taken by **finite differences**; the **strategy**, a 0-to-3 knob, trades function calls for their accuracy



Later today, 16:00 CET: J. Bendavid at the PHYSTAT seminar in memory of Fred James, from MINUIT to auto-differentiation – RooFit and Clad play a key role, and the lineage is from his slides.

A Minuit 2 minimization in three stages

1 · Start point
Minuit's "seeding"

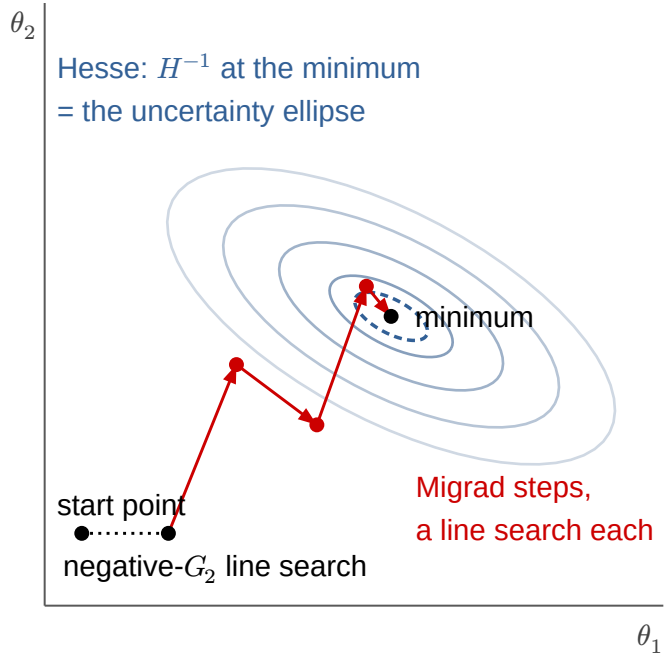
- f and ∇f at the start point
- $G_2 = \partial^2 f / \partial \theta_i^2$, the diagonal of H (Minuit's name for it), for the first error matrix $V_0 = \mathrm{diag}(1/G_2)$
- some $G_2 \leq 0$: line searches along those axes until the curvature is positive

2 · Migrad
variable-metric iterations

- ∇f at every new point
- a line search along $-V\nabla f$: function values only
- rank-2 update of V (DFP / BFGS); converged when the estimated distance to the minimum, EDM $= \frac{1}{2}\nabla f^T V \nabla f$, is below the tolerance

3 · Hesse
at the minimum

- the full H by finite differences of f, or from the function if it provides one
- $V = H^{-1}$: covariance, errors, correlations – **the result of the measurement**
- strategy 1: only if the running estimate drifted; 2 and 3: always



● analytical, from Clad today ● numerical, even with an analytical gradient ● analytical, but opt-in ● no derivative

What still falls back to numerical differentiation

To never fall back, Minuit 2 needs from the likelihood:

1. G_2 **at the start point**: the Hessian diagonal there, for the first error matrix – why not $V_0 = I$: parameters span orders of magnitude, and the EDM stopping rule uses V, so a bad V_0 is wasted iterations
2. G_2 **after each negative-**G_2 **correction** – same derivative, a few more points
3. **Hesse on by default** once the analytical path honours the sparsity mask and does not force positive-definiteness (and treats bounded parameters right: a curvature term of their transform was missing until June 2026)

The cost today: on Combine's 2487-parameter fit the start point's G_2 is 58 of 83 s (part 3).

Interfaces exist for all of it (`HasG2`, `HasHessian`, `SecondDerivativeAlwaysVanishes`); what is missing are **cheap** second derivatives from the AD side: an exact diagonal from N second-order passes (`clad::hessian_diagonal`) costs about what the finite differences cost – cheap means **coloured Hessian-vector products** (part 4).

Part 2 – RooFit: Clad in the likelihood framework

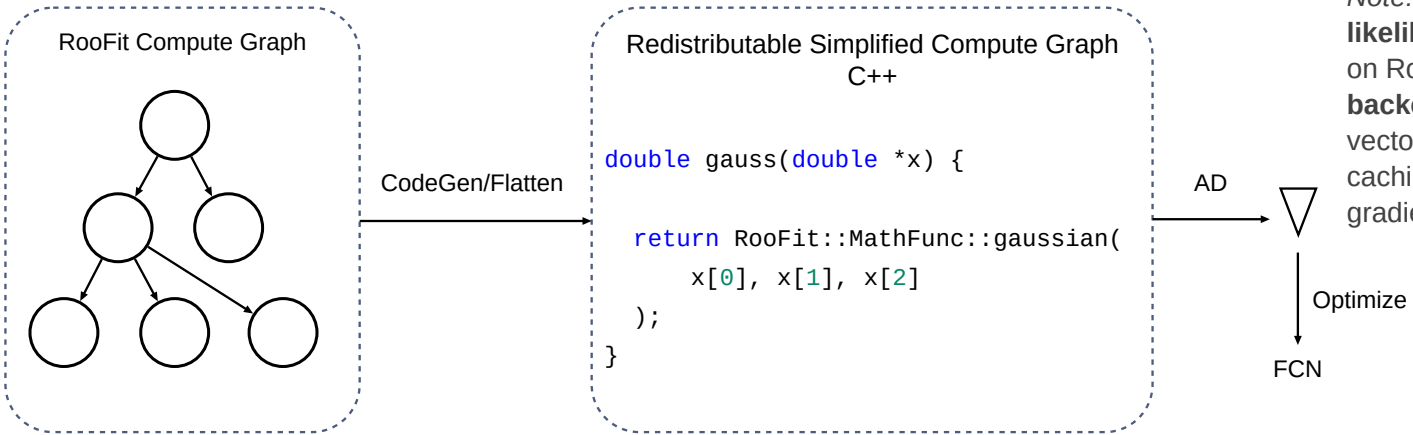
Automatic differentiation with Clad

RooFit uses Clad to get analytic gradients: **Code generation** (aka. “codegen”)

More detail in [ROOT March 2025 blog post]

1. **Mathematical** concept
2. **RooFit** user code
3. **Automatic translation** of RooFit model to simple C++ code
4. **Gradient** of C++ code **automatically generated** with **Clad**
5. Gradient code **wrapped** back into RooFit object

Shown in June 2025; new since: Hessians, and the shapes Clad had to learn (next page). Details: [RUW 2025 talk].



Note: the **nominal likelihood** still runs on RooFit’s **CPU backend**, for its vectorization and caching outside the gradients.

What Clad gets to see

- RooFit translates the computation graph into **plain C++**: one function per **channel** (one histogram of the analysis; channels share parameters) over flat arrays (parameters, data, constants)
- The maths lives in **free functions** with custom pullbacks where Clad needs help – and now pushforwards, for the Hessians
- The generated code is text: **inspectable**, and dumpable as a standalone macro – the reproducer behind every Clad issue filed from RooFit
- Clad sees checkpointed loops, pointer arithmetic over `const` data, nested calls: the shapes its Hessian mode had to learn

C++

```
// generated by RooFit for a binned likelihood
double nll_channel_0(double *params,
                     double const *obs,
                     double const *xlArr) {
  double result = 0.;
  #pragma clad checkpoint loop
  for (int bin = 0; bin < 128; ++bin) {
    double expected = xlArr[bin]
      * flexibleInterp(params + 7,
                       xlArr + 128 + 36 * bin, 18)
      * params[bin + 25];        // per-bin gamma
    result += poissonNll(obs[bin], expected);
  }
  return result + gaussianConstraints(params, xlArr);
}

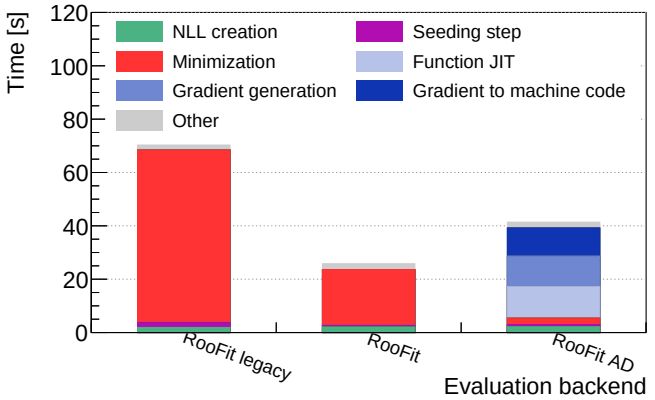
// ... one function per channel, summed up ...
#pragma clad ON
void nll_grad_req() { clad::gradient(nll, "params"); }
void nll_hess_req() { clad::hessian(nll, "params[0:152]"); }
#pragma clad OFF
```

A sketch: the real functions carry the variable names and node types of the model in comments.

The ATLAS Higgs combination benchmark

- Using analytic gradients significantly **reduces minimization time** for many-parameter fits with
  - ATLAS HistFactory benchmark on the right
- When using AD, about 50 % of the minimization is **gradient evaluation**, the rest is the **line search**
- The same gain on a current CMS analysis: a 100-point likelihood scan of a boosted H → cc model, **660 s → 260 s** (T. Runting, CHEP 2026)
- Also **numerically more stable**: no tricks required to get better precision on numeric gradients *(e.g. likelihood offsetting)*
- *Caveat:* potentially long time for gradient generation
  - To benefit, workflow needs to reuse likelihood (e.g. **toy studies** or **profile likelihood scans**)

ATLAS HistFactory fit *ROOT dev, Feb 2026*



pre-ROOT 6.32
(before 2024)

Detailed breakdown of minimization time for ATLAS Higgs combination benchmark with different RooFit backends (49 HistFactory channels, 739 parameters in total, in rootbench).

Updated version of a result from ICHEP 2024.

Hessians in RooFit: the road to Clad 2.5

- **Feb 2026**: RooFit plumbs `clad::hessian` through to Minuit 2 (ROOT #21416) – “it doesn’t work yet”: `#pragma clad checkpoint loop` broke it (clad #1736)
- **Clad 2.3** (Apr): nested Hessians fixed → **ROOT 6.40** (May) is the first release with working Clad Hessians, opt-in
- **Spring and summer**: the long tail from real likelihoods – Hessians **silently all zero** (`double const *` in nested functions), **every multi-channel Hessian crashing** (pointer arithmetic on a null tangent), …
- **Aug–Sep 2026**: 36 commits from the RooFit side in four weeks, merged by Vassil within days; **Clad 2.5** (28 Sep) closes 50 issues, **ROOT pins it the same day**

Clad pull requests, Aug–Sep 2026

#1993
Hessians from Hessian-vector products

#1988
the constant folder in forward mode

#1983
TBR on the Hessian’s inner reverse pass

#1979
`nullptr + 1` passing the null check

The user’s function is the test suite: nothing here was found on a toy.

How Clad builds a Hessian since 2.5

- **Before**: one forward-mode derivative per parameter, each reverse-differentiated → code grew fast with N: 242 k lines of IR at 64 parameters
- **Now**: $Hv = \nabla(\nabla f \cdot v)$ – a pullback of a pushforward, seeded with $(0, 1)$; the wrapper loops over the unit vectors – part 4 asks to own that loop
  - code $O(|f|)$, run time N pullback calls of ≈ 2 to 3 gradient evaluations each
- 81-parameter HistFactory likelihood (binned templates with systematic variations, the LHC standard): Hessian generation **64.8 s → 2.2 s**, identical minima

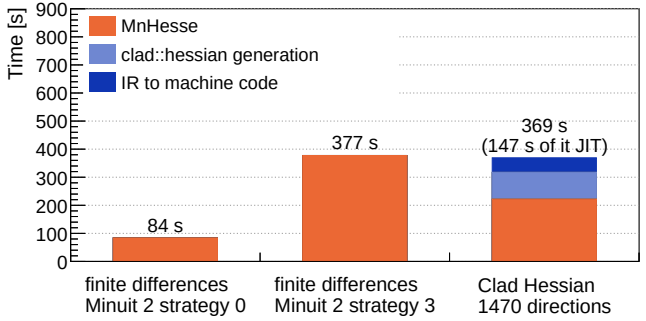
C++, generated

```
void nll_hessian_0(double *params, double const *obs,
                   double const *xlArr, double *hessian) {
  clad::ValueAndPushforward<double, double> _d_y{0., 0.};
  _d_y.pushforward = 1.;    // seed: pull back (grad f . v)
  double _d_params[N]{0};   // the direction v
  _d_params[0] = 1.;        // v = e_0
  nll_pushforward_0_pullback(params, obs, xlArr,
      _d_params, nullptr, nullptr, _d_y,
      hessian + 0 * N);     // row 0 = H e_0
  _d_params[0] = 0.;
  _d_params[1] = 1.;        // v = e_1
  nll_pushforward_0_pullback(params, obs, xlArr,
      _d_params, nullptr, nullptr, _d_y,
      hessian + 1 * N);
  _d_params[1] = 0.;
  // ... N calls of the same pullback
}
```

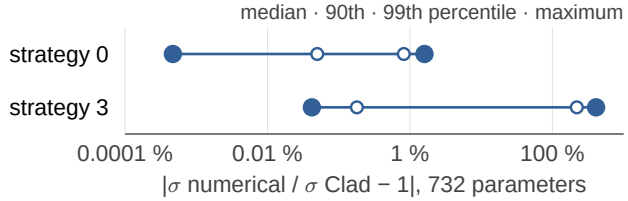
The wrapper Clad 2.5 emits, in the shape RooFit's likelihood gets it.

Hesse on the ATLAS benchmark: numerical vs Clad

- **Accuracy**: strategy 0 is within 1.6 % here, but **forced positive-definite** – the distortion that goes wrong on fits to large datasets; strategy 3 (table in the backup) cures that and is in turn up to **5× off** on the flattest directions, per-bin parameters of the simulation samples – with AD there is no step, cycle or tolerance to get wrong
- **Time**: strategy 0 in 84 s, strategy 3 in 377 s; Clad: 147 s of JIT once per process, then **222 s** per evaluation
- **Not faster than strategy 0 yet**: 1470 pullbacks of 0.15 s (two to three gradients' worth each), half of them for parameters constant in this fit – next: floating parameters only (2×), colouring, vector mode
- The **sparsity mask** (#16394, Sep 2026) is already in: numerical Hesse skips the pairs that never share a term, a quarter of its time; the 84 s above uses it



The ATLAS benchmark of page 12, rootbench's VH(bb) workspace (the RooFit file holding model and data): 1470 parameters, 732 floating. Migrad at strategy 0, then `RooMinimizer::hesse()`; one process per bar, one core, best of two runs; #21622 *with Clad 2.5, run 2026-11-03.*



Part 3 – CMS Combine: a full AD pipeline

Combine, the statistical analysis tool of CMS

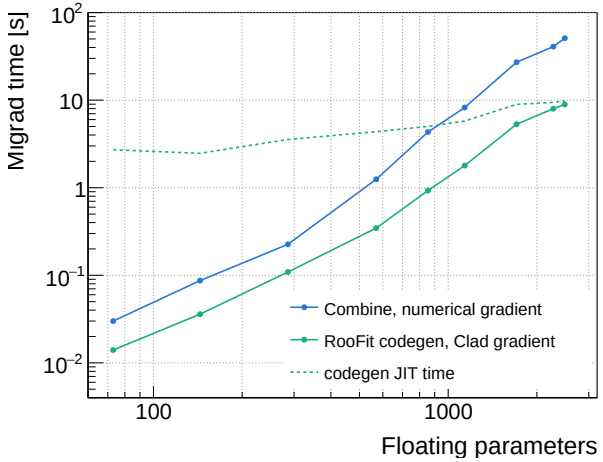
Part 2 was one benchmark; Combine is where the users are.

- Builds the RooFit likelihood from a text **datacard** and runs every statistical procedure on it; used by **the majority of CMS measurements and searches** – the largest user base of any experiment-specific fit framework – and, through about 30 published likelihoods, outside CMS
- The scale: the Run 2 (2015–18 data) single-Higgs combination has **thousands of parameters**; individual fits take up to **about 22 hours**, the Run 1 (2010–12) model two minutes; run as **grid jobs**, and with numerical gradients **about 90 % of a fit is the gradient**
- A faster fit is CPU hours saved and a working day given back to the physicist
- Built with CMS in the **hackathon weeks** of its Common Analysis Tools group (three since the last talk; the 8th, Oct 2026: **36 pull requests merged**; timeline in the backup); Compiler Research's P. Zarytskyi set up Combine's coverage CI

Paper: *The CMS statistical analysis and combination tool: Combine* (2024).

CMSHistSum: most CMS fits can now run with AD

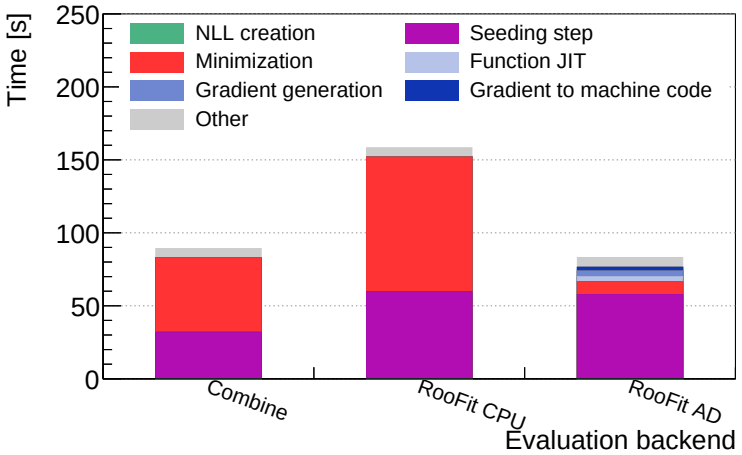
- `CMSHistSum`, Combine's **optimized binned likelihood**: its thousands of per-bin parameters for the finite simulation samples never reach Minuit, **closed-form** profiling (Barlow–Beeston "lite")
- ⎇ #1331 (Oct 2026): codegen for `CMSHistSum` through **one kernel**, the histogram templates baked in as constants, independent of the histogram size
  - no differentiation through the inner per-bin solve: by the **envelope theorem** ($\partial L/\partial\gamma = 0$ there) the gradient of the profiled likelihood is the partial derivative at the solution
- **Benchmark**: Combine's caching NLL (numerical gradients) against RooFit codegen (**Clad gradient**), same workspace: Migrad iterations **5.7× faster** at 2487 parameters
- The JIT (2.7 s → 9.8 s) is passed by Combine's own Migrad from about 1000 parameters: AD wins from there, JIT counted



One Migrad pass, strategy 0, single core: a CMSHistSum $H \to \tau\tau$ category replicated n times with private nuisances, $2 + 71n$ parameters; run 2026-11-03 on the ⎇ #1331 branch.

Benchmark: where the time goes at 2487 parameters

- Migrad iterations: **51 s → 9 s** with AD
- But the process is only 7 % faster: the **start point** costs 58 s, **69 % of the run** – Minuit 2 computes G_2 by finite differences no matter what gradient it was given, and RooFit's evaluator runs the whole likelihood per perturbed parameter where Combine's caching NLL runs one channel (32 s on its bar)
- Back to part 1: the **start point's second derivatives** are the biggest lever left – a few coloured Hessian-vector products (part 4) would replace those 58 s by a few pullbacks

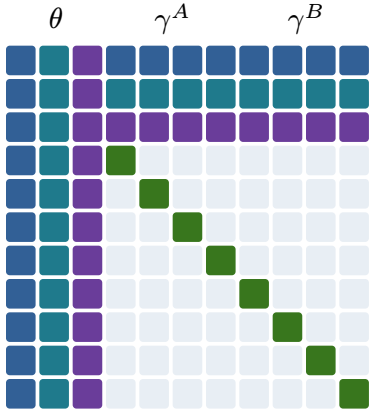


The whole `combine` process for the last point of the scaling plot: 89 s, 158 s and 83 s. The seeding step is the start point: the numerical gradient and G_2 there, plus the negative-G_2 line searches.

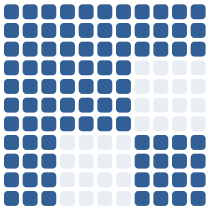
Part 4 – Future directions and a wishlist for Clad

Sparsity: Hessian-vector products seeded by colour

- RooFit already tells Minuit 2 which parameter pairs never share a term (the mask of page 15); the classic next step is **colouring**: perturb mutually independent parameters at once
- Its **exact AD analogue**: seed one Hessian-vector product with the sum of a colour class's unit vectors – outside the shared rows those columns are disjoint, so one pullback returns them all
  - shared parameters + 1 direction instead of N, and the start point's G_2 for free



The true structure, by seed: per-bin parameters couple only to θ and themselves, so 3 + 1 directions instead of 11.



What RooFit's mask knows today: shared θ couple to everything, a channel's γ to their own channel.

The case for a Hessian-vector-product API in Clad

- rabbit **never forms the Hessian**: its trust-region Krylov minimizer needs only **Hessian-vector products** (HVPs), $Hv = \nabla(\nabla f \cdot v)$, each a few gradient evaluations, independent of N – the matrix would be 80 GB at $N = 10^5$
- No positive-definite forcing: negative curvature just sends the step to the boundary. The W-mass fit in under 40 s; 100 000 bins × 8192 parameters in under 10 minutes, the HVPs dominating
- RooFit wants the same primitive, so that **no further fit framework** has to be built: the start point's G_2 and Hesse from coloured HVPs, impacts, an HVP-driven minimizer
- **Clad 2.5 already generates it**: the pullback of a pushforward inside the Hessian wrapper *is* Hv – but the wrapper calls it N times with $v = e_i$. **The ask**: a public entry point that takes the direction v, and vector mode over a batch of directions

Clad 2.5 today: `clad::hessian(f)`

```
for i in 0..N: v = e_i
  row_i = pullback(pushforward(f), v)
```

one inner pullback, N calls, not public

↓ make it public, take any v

The ask: `clad::hvp(f, v)`

one pullback for any direction v: Hv at a few gradients' cost

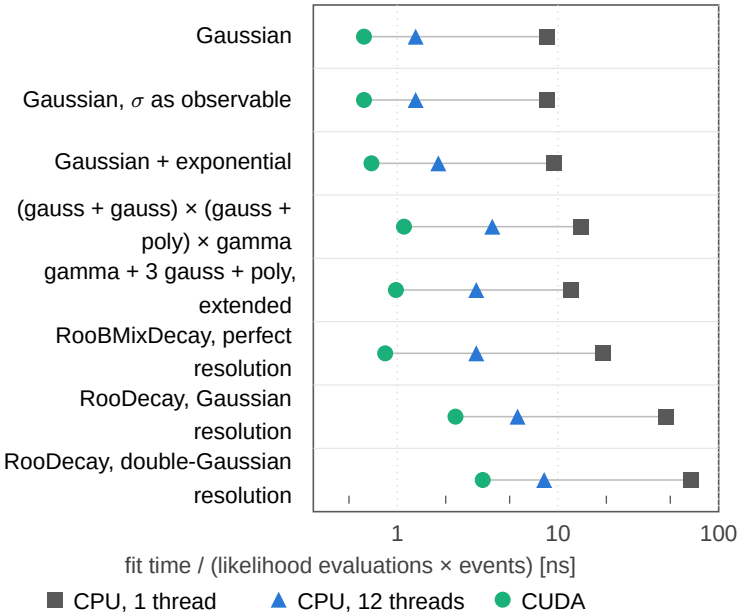
vector mode over a batch of directions

The numbers are rabbit's (D. Walter, J. Bendavid, K. Long; arXiv:2608.26376): TensorFlow 2 + SciPy, written when RooFit + Minuit was found insufficient for the CMS W-mass fit, a 3D histogram with 2880 bins and 4833 nuisance parameters.

GPUs need AD: the FP64 wall

- The CUDA backend is 10–20× faster than one CPU thread on unbinned fits, but only 2–4× faster than twelve: it is bound by **FP64 throughput**, which NVIDIA's data-centre parts are cutting
- **Numerical derivatives need FP64**: small differences of large sums. AD does not: FP32 per-event terms, FP64 accumulation

AD is what unblocks the GPU for likelihood fits, so Clad's GPU work – differentiating CUDA kernels – is **very valuable for RooFit**.



RooFit's evaluation backends on unbinned fits, 1 M events each, ROOT 6.41.01, Ryzen 9 5950X vs RTX A4500. Redrawn from D. Piparo, ROOT and GPUs, WLCG/HSF workshop, 3 Nov 2026.

Next steps, and where you can plug in

Asks to Clad, each with the page that needs it:

1. **A Hessian-vector-product entry point** that takes the direction v – the pullback of the pushforward Clad 2.5 already emits inside the Hessian wrapper (p. 14), made public; first step: a public `clad::pullback` (#1754). For the start point and Hesse (pp. 19, 22)
2. **Vector mode over a batch of directions**, with custom seeds: the coloured products of p. 21 give the start point's G_2 and Hesse in a few calls
3. **One planning pass** for gradient, G_2 and Hessian of one function – shared pushforwards and pullbacks, no dead code: the JIT time of pp. 15 and 18
4. **CUDA kernels**: differentiating single-precision likelihoods for the GPU backend (p. 23)

Where to start: #2002 (a HistFactory-style likelihood generator for the benchmarks), `test/Hessian/` (the RooFit-shaped regression tests), `-Rclad-perf` (where a hand-written derivative would pay).

On the ROOT and Combine side: Minuit 2 consumes analytical derivatives everywhere (the start point's G_2, the negative-G_2 corrections, Hesse by default), the second-order routes measured on the Higgs combinations; Combine merges #1331 and flips the default backend for binned models – the goal is the Run 2 Higgs combination with AD.

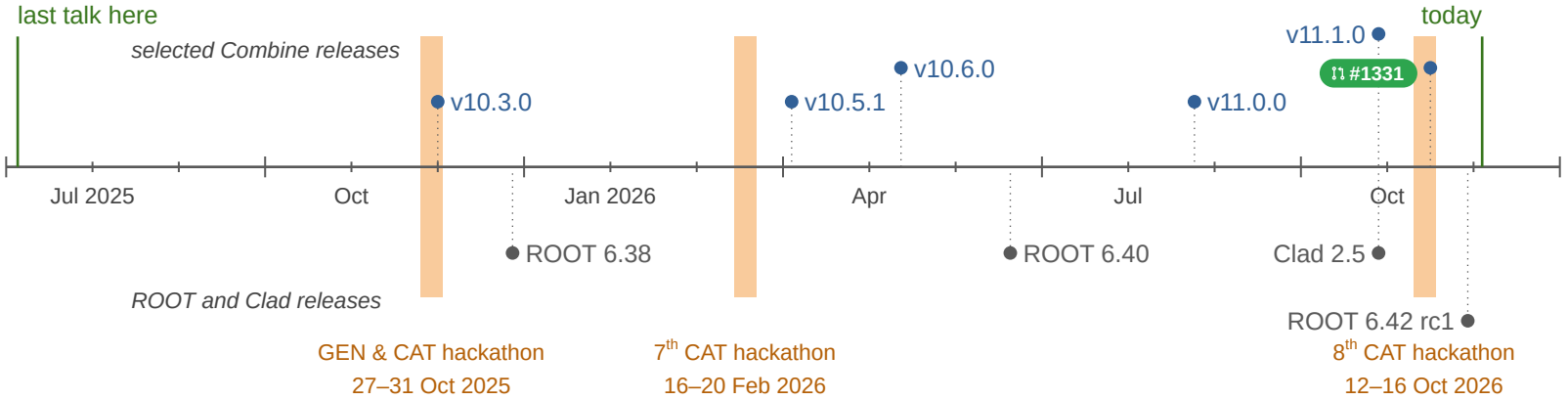
Conclusions

- Clad is the **derivative engine of the LHC's likelihood stack**: ROOT ships it, RooFit generates gradients and Hessians with it, Combine is about to take it to most CMS analyses
- What made it work: **releases on both sides** (six Clad releases since June 2025, ROOT pinning each), reproducers from real likelihoods, hackathons with CMS
- **Clad 2.5**, the highlight of the year: Hessians correct on every model in the test suite, generated 30× faster
- The **start point's G_2 is the last numerical derivative** in the loop; second order is the next frontier, and it is Clad work – first step: an HVP entry point (#1754, a public pullback)

Thanks to Vassil Vassilev, Petro Zarytskyi and the Compiler Research group, and to Tom Runting, Massimiliano Galli and Aliya Nigamova of CMS Combine.

Backup

Working with CMS: the hackathon timeline



- The CMS **Common Analysis Tools** group runs a few hackathon weeks a year; in between Combine stabilizes, which earns the next radical change
- The 8th hackathon, 12–16 October 2026: **36 pull requests merged**, 13 issues closed, test coverage 20 → 30 %
  - on the AD path: **channel masking and summed Clad gradients** #1318, a **custom Clad pullback** for `RooPower` #1303, `CMSHistSum` slimmed for the port #1324
- It goes both ways: Combine developers land RooFit features in ROOT; Compiler Research's P. Zarytskyi set up Combine's coverage CI

What Minuit 2 asks of the likelihood, and who answers

Stage	Needs	From Clad today (RooFit codegen)	Otherwise
Start point	f, ∇f at the start point	🟢 external gradient (`FCNBase::Gradient`)	$2N$ evaluations per cycle
	G_2, the diagonal of H, for V_0	🔴 RooFit has no `G2()`; given a Hessian, Minuit 2 evaluates the **full** H and keeps its diagonal	$2N$ evaluations per cycle, **plus a numerical gradient** despite the analytical one
	negative-G_2 line search	🔴 G_2 again: one more full H per corrected parameter	G_2 falls out of the numerical gradient
Migrad	∇f per iteration; the line search needs f only	🟢 external gradient, every iteration	$2N$ evaluations per cycle
Hesse	the full H	🟡 analytical if the function has gradient **and** Hessian (`useHessian`, off by default); forced positive-definite, sparsity mask ignored	$N(N-1)/2$ entries plus diagonal cycles; with only an external gradient, Hesse **recomputes a numerical one** for its step sizes
Minos	f, plus Migrad at a lower strategy	🟢 through the gradient	–

The gradient is solved; **second-order** questions still end in finite differences unless the likelihood opts in.

Strategies: trading function calls for reliability

tunable (`MnStrategy`)	0	1 (default)	2	3
full Hessian at the start point	–	–	yes	–
gradient: cycles / step tolerance / tolerance	2 / 0.5 / 0.1	3 / 0.3 / 0.05	5 / 0.1 / 0.02	5 / 0.1 / 0.02
Hesse diagonal: cycles / step tolerance / G_2 tolerance	3 / 0.5 / 0.1	5 / 0.3 / 0.05	7 / 0.1 / 0.02	7 / 0 / 0
Hesse: gradient refinement cycles	1	2	6	6
Hesse: central differences for mixed derivatives	–	–	–	yes (4 calls each)
Hesse: force positive-definite	yes	yes	yes	no
Hesse after Migrad	never	if the estimate drifted	always	always

- Every row is a **hyperparameter of numerical differentiation**
- Strategy 3 (W. Buttinger, 2023): the cure for the Hessians of high-statistics fits going wrong
- Its “no” to forcing positive-definiteness holds for an explicit `hesse()`; the Hesse that Migrad runs itself always forces it
- With analytical derivatives the table collapses: strategies 0 and 3 give the same Hessian