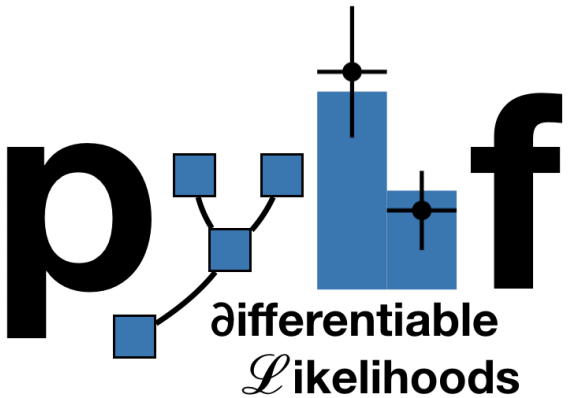
# pyhf

a pure Python statistical fitting library with tensors and autograd




Matthew Feickert

matthew.feickert@cern.ch
@HEPfeickert
matthewfeickert

SciPy 2020
July 7th, 2020

# `pyhf` core dev team

Lukas Heinrich

CERN

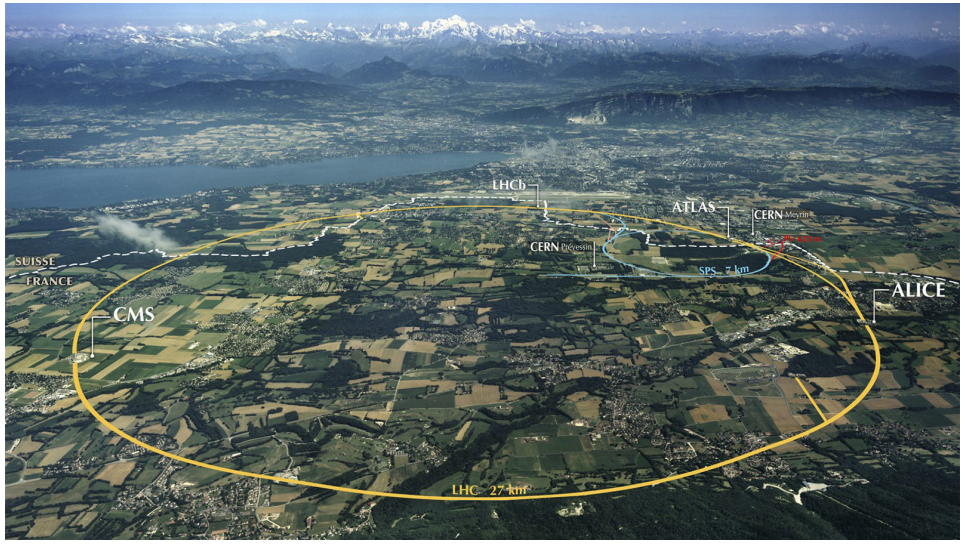
Matthew Feickert

University of Illinois
Urbana-Champaign

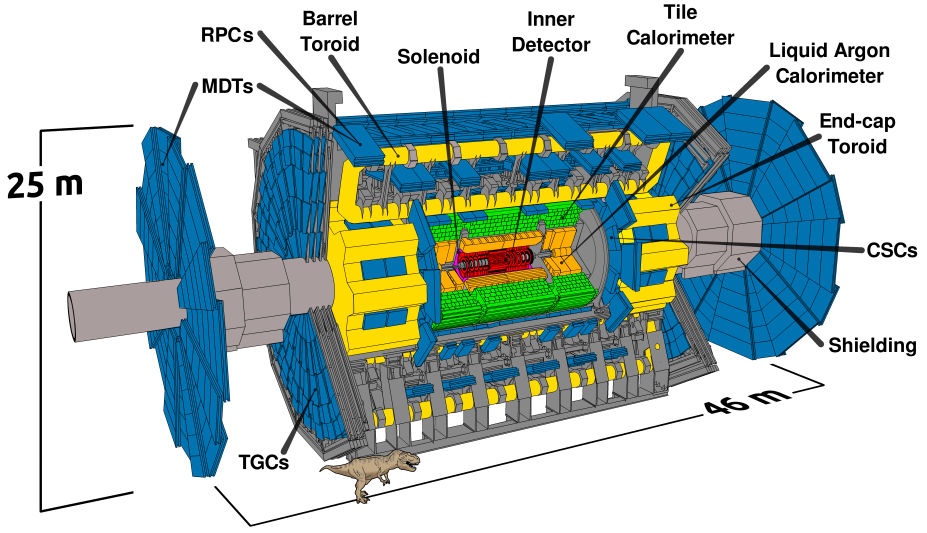
Giordon Stark

UCSC SCIPP

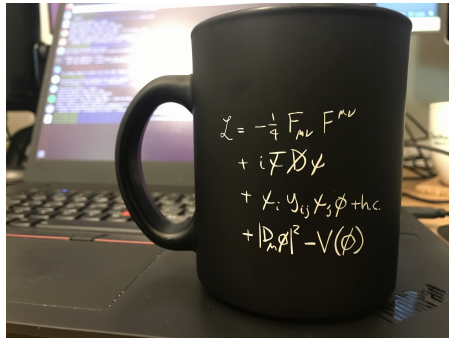
# We're high energy particle physicists




LHC



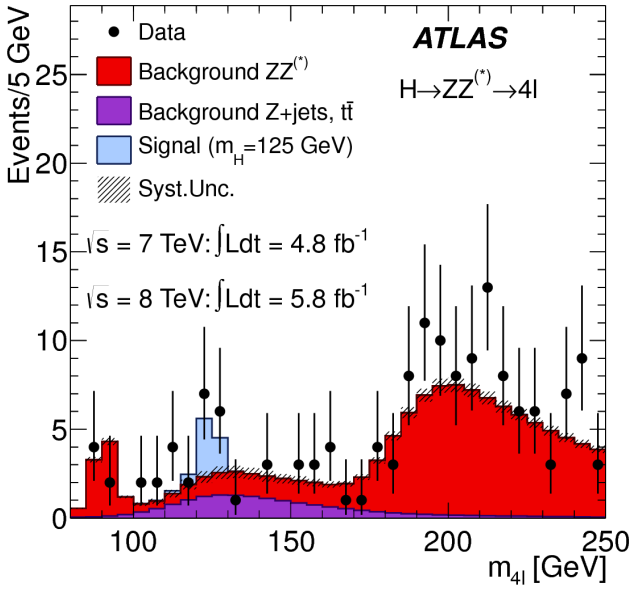

ATLAS



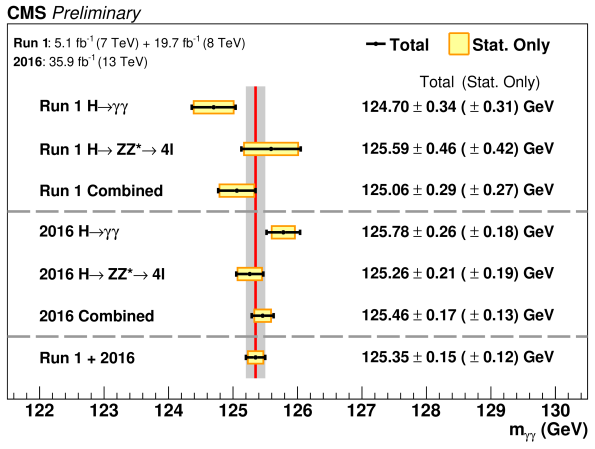


# So we want to know

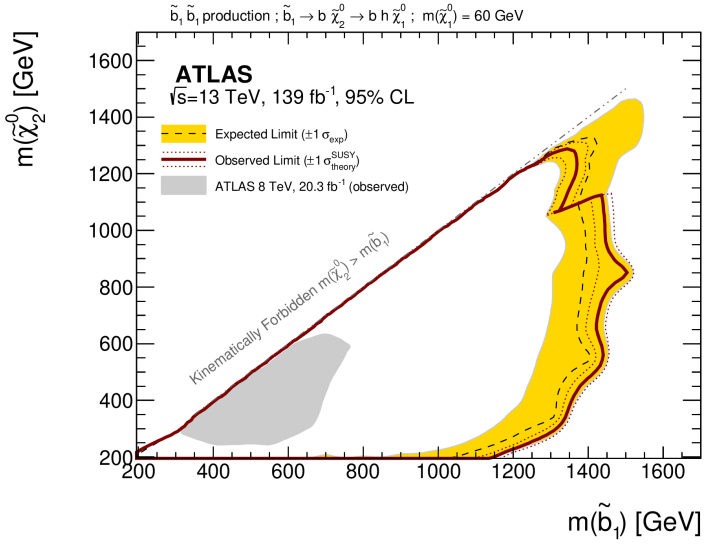
# Goals of physics analysis at the LHC




Search for new physics




Make precision measurements

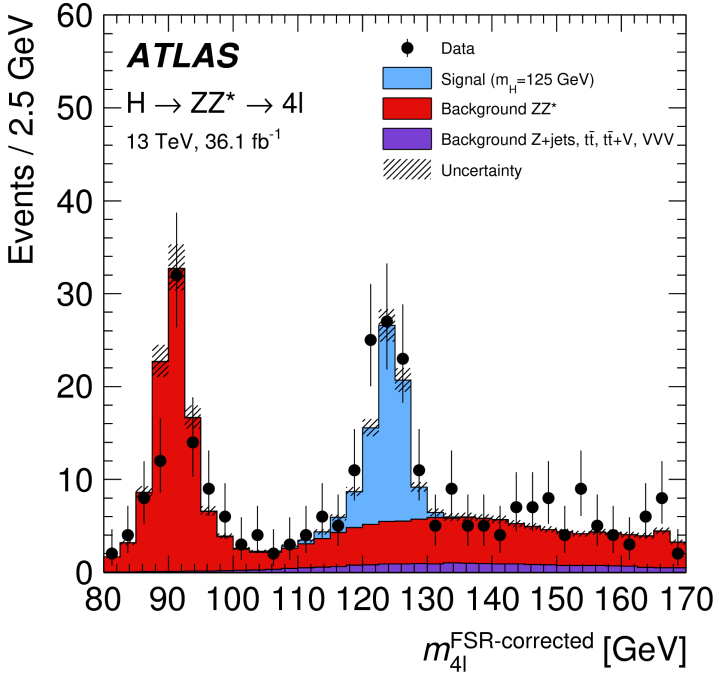



Provide constraints on models through setting best limits

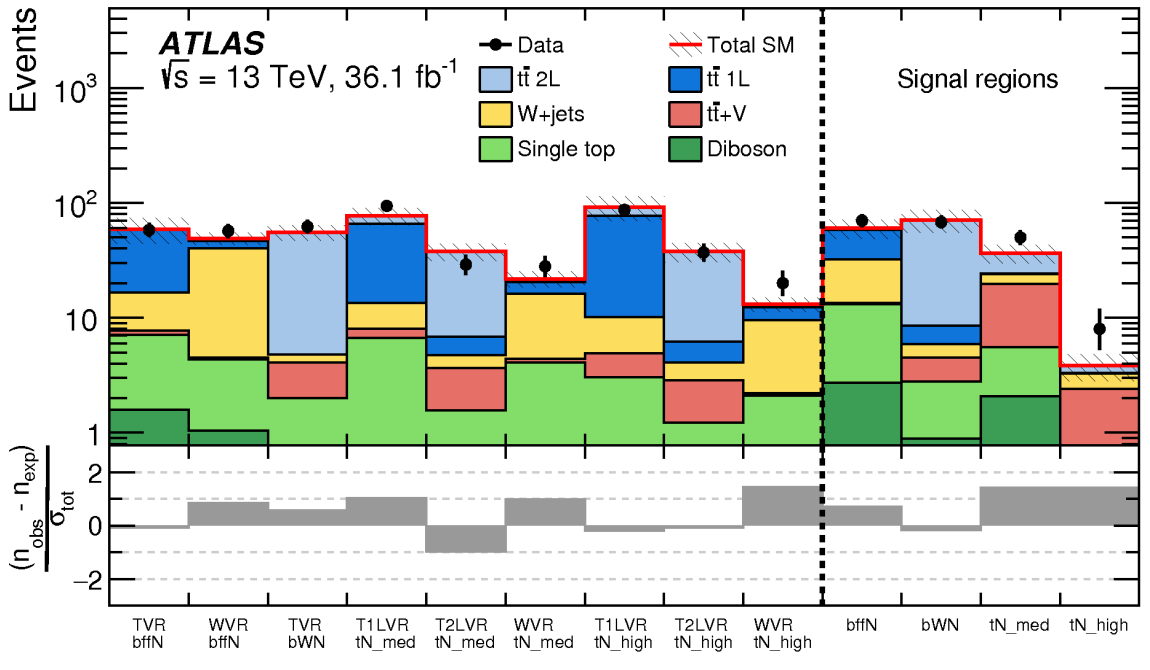
- All require **building statistical models** and **fitting models** to data to perform statistical inference
- Model complexity can be huge for complicated searches
- Problem: Time to fit can be **many hours**
- Goal: Empower analysts with fast fits and expressive models

# HistFactory Model

- A flexible probability density function (p.d.f.) template to build statistical models in high energy physics (HEP)
- Developed during work that lead to the Higgs discovery in 2011 [CERN-OPEN-2012-016]
- Widely used by the HEP community for **measurements of known physics** (Standard Model) and **searches for new physics** (beyond the Standard Model)




**Standard Model**




**Beyond the Standard Model**

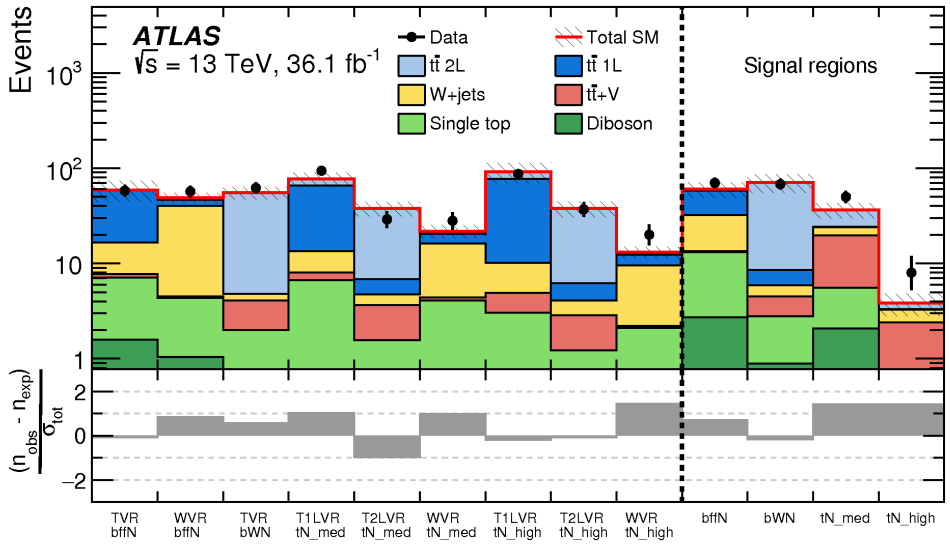
# HistFactory Template

$$f\left(\text{data}|\text{parameters}\right) = f\left(\vec{n},\vec{a}|\vec{\eta},\vec{\chi}\right) = \prod_{c\in\text{channels}}\prod_{b\in\text{bins}_c}\text{Pois}\left(n_{cb}|\nu_{cb}\left(\vec{\eta},\vec{\chi}\right)\right)\prod_{\chi\in\vec{\chi}}c_\chi\left(a_\chi|\chi\right)$$

**Use:** Multiple disjoint channels (or regions) of binned distributions with multiple samples contributing to each with additional (possibly shared) systematics between sample estimates

**Main pieces:**

- Main Poisson p.d.f. for simultaneous measurement of multiple channels
- Event rates $\nu_{cb}$ (nominal rate $\nu^0_{scb}$ with rate modifiers)
- Constraint p.d.f. (+ data) for "auxiliary measurements"
  - encode systematic uncertainties (e.g. normalization, shape)
- $\vec{n}$: events, $\vec{a}$: auxiliary data, $\vec{\eta}$: unconstrained pars, $\vec{\chi}$: constrained pars




Example: **Each bin** is separate (1-bin) channel, each **histogram** (color) is a sample and share a **normalization systematic** uncertainty

# HistFactory Template

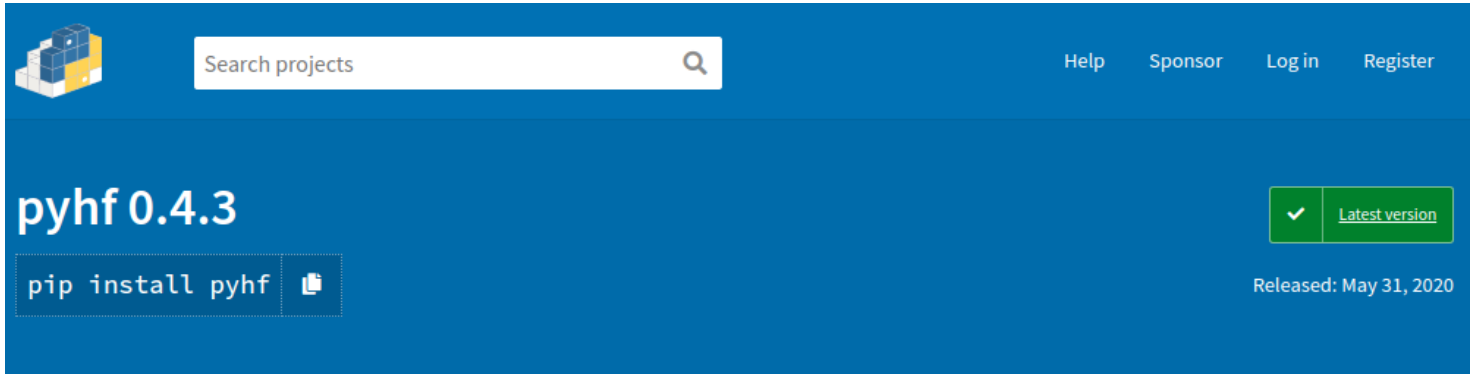
$$f\left(\vec{n}, \vec{a} | \vec{\eta}, \vec{\chi}\right) = \prod_{c \in \text{channels}} \prod_{b \in \text{bins}_c} \text{Pois}\left(n_{cb} | \nu_{cb}\left(\vec{\eta}, \vec{\chi}\right)\right) \prod_{\chi \in \vec{\chi}} c_\chi\left(a_\chi | \chi\right)$$

Mathematical grammar for a simultaneous fit with

- multiple "channels" (analysis regions, (stacks of) histograms)
- each region can have multiple bins
- coupled to a set of constraint terms

**This is a mathematical representation!** Nowhere is any software spec defined
**Until now** (2018), the only implementation of HistFactory has been in a monolithic `C++` library used in HEP (`ROOT`)

**`pyhf`: HistFactory in pure Python**



# Basic object of HistFactory is the statistical model

$$f\left(\vec{n}, \vec{a} | \vec{\eta}, \vec{\chi}\right) = \prod_{c \in \text{channels}} \prod_{b \in \text{bins}_c} \text{Pois}\left(n_{cb} | \nu_{cb}\left(\vec{\eta}, \vec{\chi}\right)\right) \prod_{\chi \in \vec{\chi}} c_\chi\left(a_\chi | \chi\right)$$

care about log likelihood as using maximum likelihood fits

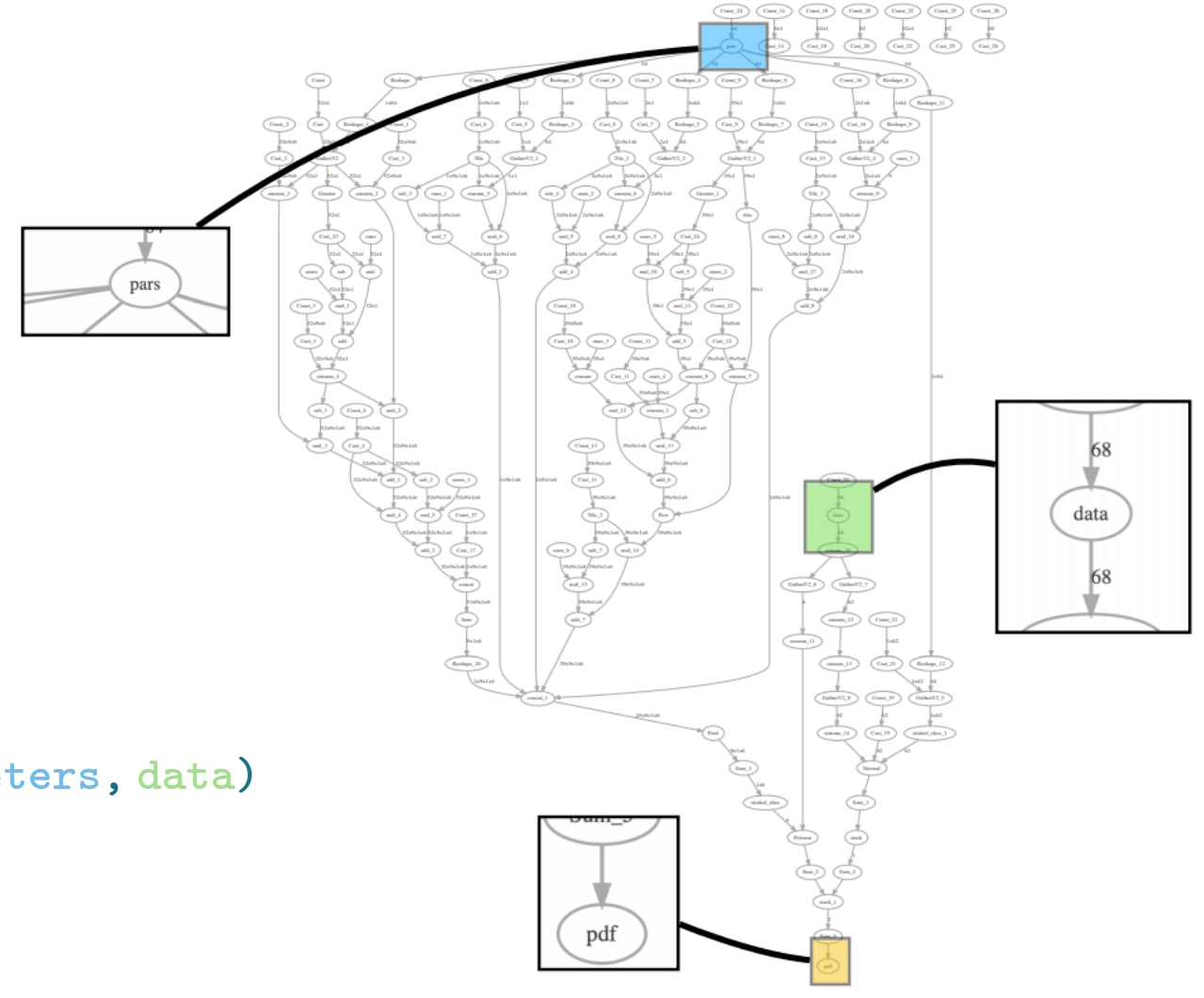
$$\ln L\left(\vec{\theta} \middle| \vec{x}\right) \Rightarrow \texttt{model.logpdf(parameters, data)}$$

```
>>> import pyhf
>>> model = pyhf.simplemodels.hepdata_like(
...     signal_data=[12.0, 11.0], bkg_data=[50.0, 52.0], bkg_uncerts=[3.0, 7.0]
... )
>>> observations = [51, 48]
>>> data = observations + model.config.auxdata
>>> parameters = model.config.suggested_init()  # nominal parameters
>>> model.logpdf(parameters, data)
array([-15.38762717])
```

# Model is represented as a computational graph

- Each node of the graph represents a vectorized $n$-dimensional array ("tensorized") operation
- The graph (model) is largely factorized between the parameter graph and the data graph
- The bottom node is then used for final log likelihood value

```
value = model.logpdf(parameters, data)
```



# Core task: Maximum likelihood fits

`pyhf.infer.mle.fit(data, model)`

minimizes the objective function $-2\ln L\left(\vec{\theta}\middle|\vec{x}\right)$ (maximizing the likelihood) with the backend's optimizer

```
>>> import pyhf
>>> model = pyhf.simplemodels.hepdata_like(
...     signal_data=[12.0, 11.0], bkg_data=[50.0, 52.0], bkg_uncerts=[3.0, 7.0]
... )
>>> observations = [51, 48]
>>> data = observations + model.config.auxdata
>>> bestfit_pars, twice_nll = pyhf.infer.mle.fit(data, model, return_fitted_val=True)
>>> model.logpdf(bestfit_pars, data)
array([-12.49196761])
>>> -2 * model.logpdf(bestfit_pars, data) == twice_nll
array([ True])
```

# Use in profile likelihood fits to ask

**"Given our model and the data, did we discover new physics?"**

$$-2\ln\Lambda(\mu) = -2\ln\frac{L(\mu,\hat{\hat{\theta}})}{L(\hat{\mu},\hat{\theta})} \quad \begin{matrix} \Leftarrow \text{constrained best fit} \\ \hline \Leftarrow \text{unconstrained best fit} \end{matrix}$$

compute (**fast as possible!**) modified $p$-value (the $\mathrm{CL}_s$) for a given parameter of interest $\mu$ — hypothesis testing!

`pyhf.infer.hypotest(testpoi, data, model)`

```
>>> import pyhf
>>> model = pyhf.simplemodels.hepdata_like(
...     signal_data=[12.0, 11.0], bkg_data=[50.0, 52.0], bkg_uncerts=[3.0, 7.0]
... )
>>> observations = [51, 48]
>>> data = observations + model.config.auxdata
>>> test_poi = 1.0
>>> CLs_obs, CLs_exp_band = pyhf.infer.hypotest(
...     test_poi, data, model, return_expected_set=True
... )
>>> CLs_obs  # Observed value
array([0.05290116])
>>> CLs_exp_band.ravel()  # -2σ, -1σ, nominal, +1σ, +2σ expected value
array([0.00260641, 0.01382066, 0.06445521, 0.23526104, 0.57304182])
```

# JSON spec fully describes the HistFactory model

- Human & machine readable **declarative** statistical models
- Industry standard
  - Will be with us forever
- Parsable by every language
  - Highly portable
  - No lock in
- Versionable and easily preserved
  - JSON Schema describing HistFactory specification
  - Attractive for analysis preservation
  - Highly compressible

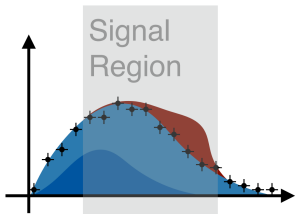
```
{
    "channels": [ # List of regions
        { "name": "singlechannel",
          "samples": [ # List of samples in region
            { "name": "signal",
              "data": [20.0, 10.0],
              # List of rate factors and/or systematic uncertainties
              "modifiers": [ { "name": "mu", "type": "normfactor", "data": null} ]
            },
            { "name": "background",
              "data": [50.0, 63.0],
              "modifiers": [ {"name": "uncorr_bkguncrt", "type": "shapesys", "data": [5.0, 12.0]} ]
            }
          ]
        }
    ],
    "observations": [ # Observed data
        { "name": "singlechannel", "data": [55.0, 62.0] }
    ],
    "measurements": [ # Parameter of interest
        { "name": "Measurement", "config": {"poi": "mu", "parameters": []} }
    ],
    "version": "1.0.0" # Version of spec standard
}
```

JSON defining a single channel, two bin counting experiment with systematics

# JSON Patch for signal model (reinterpretation)

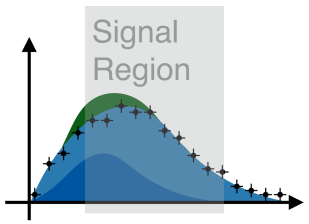
JSON Patch gives ability to **easily mutate model**

Think: test a **new theory** with a **new patch**!




Signal model A

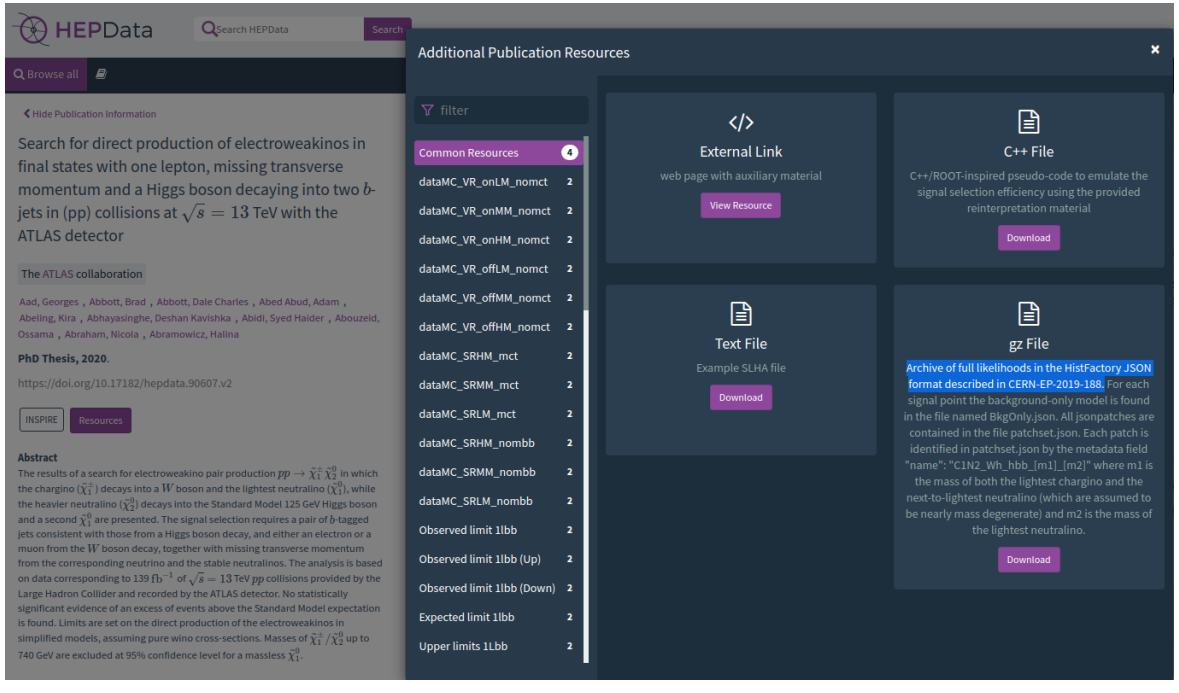
```
# Using CLI
$ pyhf cls example.json | jq .CLs_obs
0.053994246621274014

$ cat new_signal.json
[{
    "op": "replace",
    "path": "/channels/0/samples/0/data",
    "value": [10.0, 6.0]
}]

$ pyhf cls example.json --patch new_signal.json | jq .CLs_obs
0.3536906623262466
```

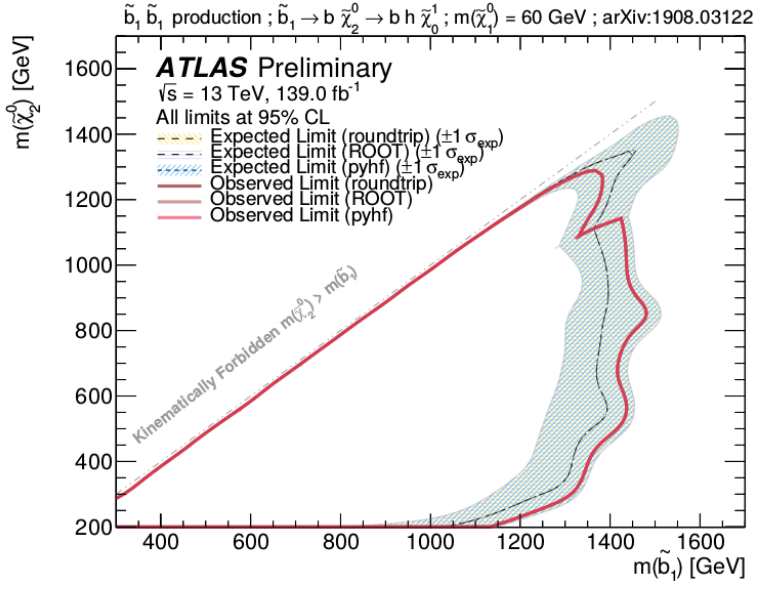



Signal model B

# Likelihood serialization and reproduction/reuse

- Background-only model JSON stored
- Hundreds of signal model JSON Patches stored together as a "patch set" file
- Together are able to publish and fully preserve the full likelihood (with own DOI! DOI 10.17182/hepdata.90607.v2/r2 )
- Shown to reproduce results but faster! **C++ (ROOT):** 10+ hours **pyhf:** < 30 minutes



# Likelihood serialization and reproduction/reuse

- Background-only model JSON stored
- Hundreds of signal model JSON Patches stored together as a "patch set" file
- Together are able to publish and fully preserve the full likelihood (with own DOI! DOI 10.17182/hepdata.90607.v2/r2 )
- First **ever** full likelihood of an LHC experiment published in 2019
  - ATLAS Run-2 search for bottom-squarks [JHEP12(2019)060]

Solves technical problem of distribution and made good on a 19(!) year old agreement to publish likelihoods

**Massimo Corradi**

It seems to me that there is a general consensus that what is really meaningful for an experiment is *likelihood*, and almost everybody would agree on the prescription that experiments should give their likelihood function for these kinds of results. Does everybody agree on this statement, to publish likelihoods?

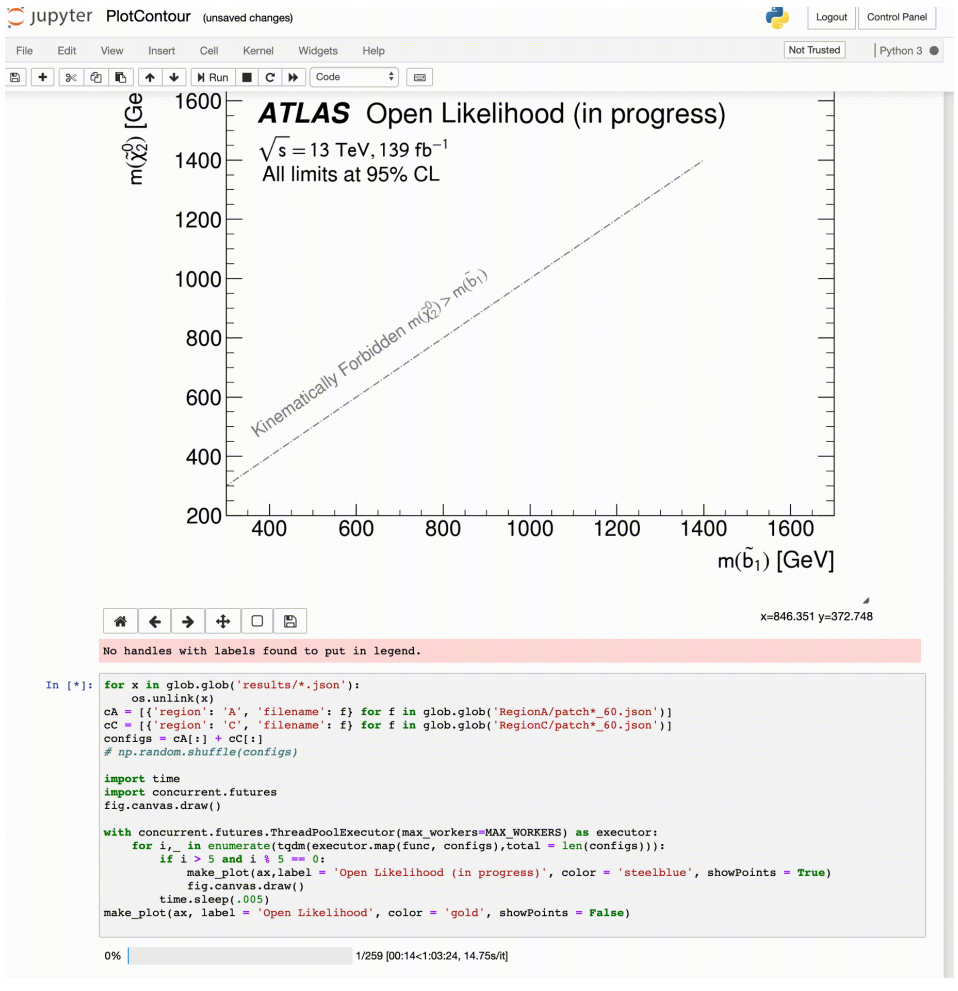
**Louis Lyons**

Any disagreement ? Carried unanimously. That's actually quite an achievement for this Workshop.

(1st Workshop on Confidence Limits, CERN, 2000)

# Further speed up by parallelizing across cluster

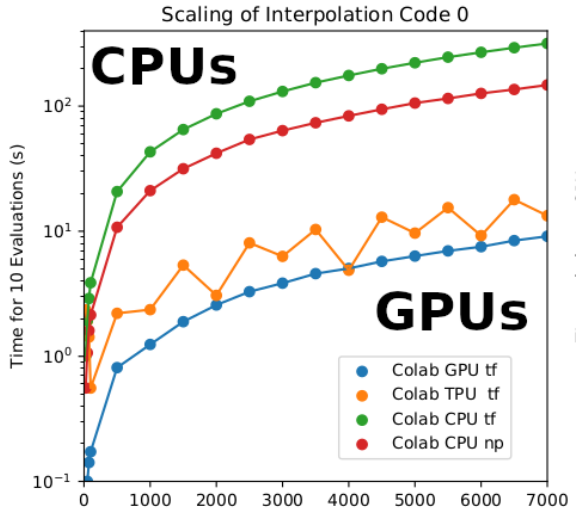
- Running across 25 worker nodes on the cloud
- Background and signal patches are sent to workers on demand
  - Possible as the signal patches don't need information from each other
  - Embarrassingly parallelizable
- Results being plotted as they are streamed back
- Fit of same likelihood now takes **3 minutes** for all signal points!




(GIF sped up by 8x)

# Machine Learning Frameworks for Computation

- All numerical operations implemented in **tensor backends** through an API of $n$-dimensional array operations
- Using deep learning frameworks as computational backends allows for **exploitation of auto differentiation (autograd) and GPU acceleration**
- As huge buy in from industry we benefit for free as these frameworks are **continually improved** by professional software engineers (physicists are not)




- Show hardware acceleration giving **order of magnitude speedup** for some models!
- Improvements over traditional
  - 10 hrs to 30 min; 20 min to 10 sec

NumPy

PyTorch

TensorFlow

# Unified API through `tensorlib` shim

```python
class _BasicNormal(object):
    def __init__(self, loc, scale):
        self.loc = loc
        self.scale = scale

    def sample(self, sample_shape):
        return norm(self.loc, self.scale).rvs(
            size=sample_shape + self.loc.shape
        )

    def log_prob(self, value):
        tensorlib = numpy_backend()
        return tensorlib.normal_logpdf(value, self.loc, self.scale)

# ...

def normal_dist(self, mu, sigma):
    return _BasicNormal(mu, sigma)
```

NumPy

```python
class _BasicNormal(object):
    def __init__(self, loc, scale):
        self.loc = loc
        self.scale = scale

    def sample(self, sample_shape):
        return norm.osp_stats.norm(self.loc, self.scale).rvs(
            size=sample_shape + self.loc.shape
        )

    def log_prob(self, value):
        tensorlib = jax_backend()
        return tensorlib.normal_logpdf(value, self.loc, self.scale)

# ...

def normal_dist(self, mu, sigma):
    return _BasicNormal(mu, sigma)
```

JAX

```python
def normal_dist(self, mu, sigma):
    return tfp.distributions.Normal(mu, sigma)
```

Tensorflow

```python
def normal_dist(self, mu, sigma):
    return torch.distributions.Normal(mu, sigma)
```

PyTorch

# Allows for transparently changing backend

```
>>> import pyhf
>>> pyhf.tensorlib.name  # default backend/optimizer is numpy/scipy
'numpy'
>>> means = [5, 8]
>>> stds = [1, 0.5]
>>> values = [4, 9]
>>> for backend in ["numpy", "tensorflow", "pytorch", "jax"]:
...     pyhf.set_backend(backend)
...     normals = pyhf.tensorlib.normal_dist(
...         pyhf.tensorlib.astensor(means), pyhf.tensorlib.astensor(stds)
...     )
...     normals.log_prob(pyhf.tensorlib.astensor(values))
...
array([-1.41893853, -2.22579135])
<tf.Tensor: shape=(2,), dtype=float32, numpy=array([-1.4189385, -2.2257915], dtype=float32)>
tensor([-1.4189, -2.2258])
DeviceArray([-1.41893853, -2.22579135], dtype=float64)
```

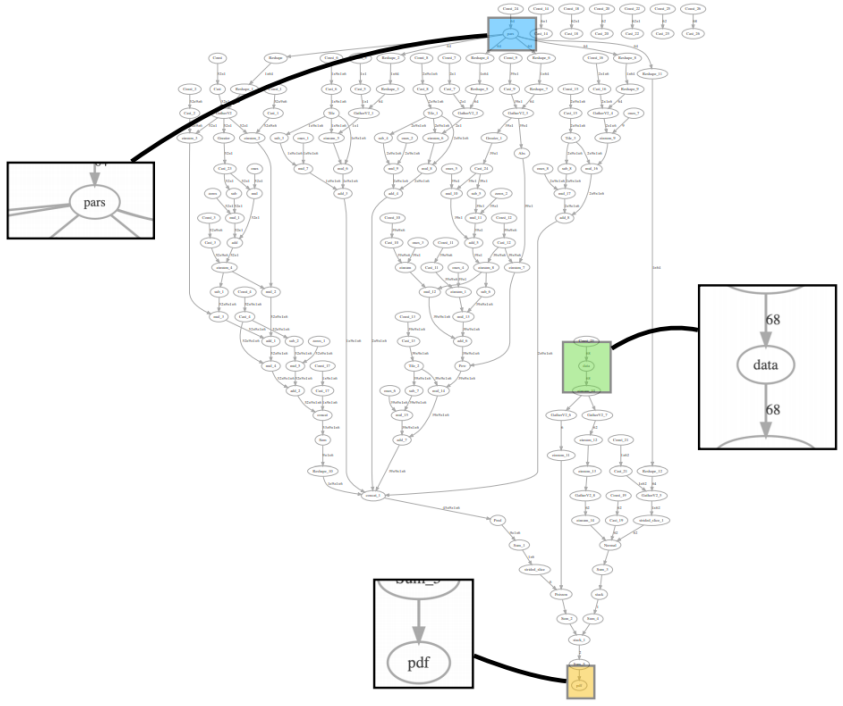
# Automatic differentiation

With tensor library backends gain access to exact (higher order) derivatives — accuracy is only limited by floating point precision

$$\frac{\partial L}{\partial \mu}, \frac{\partial L}{\partial \theta_i}$$

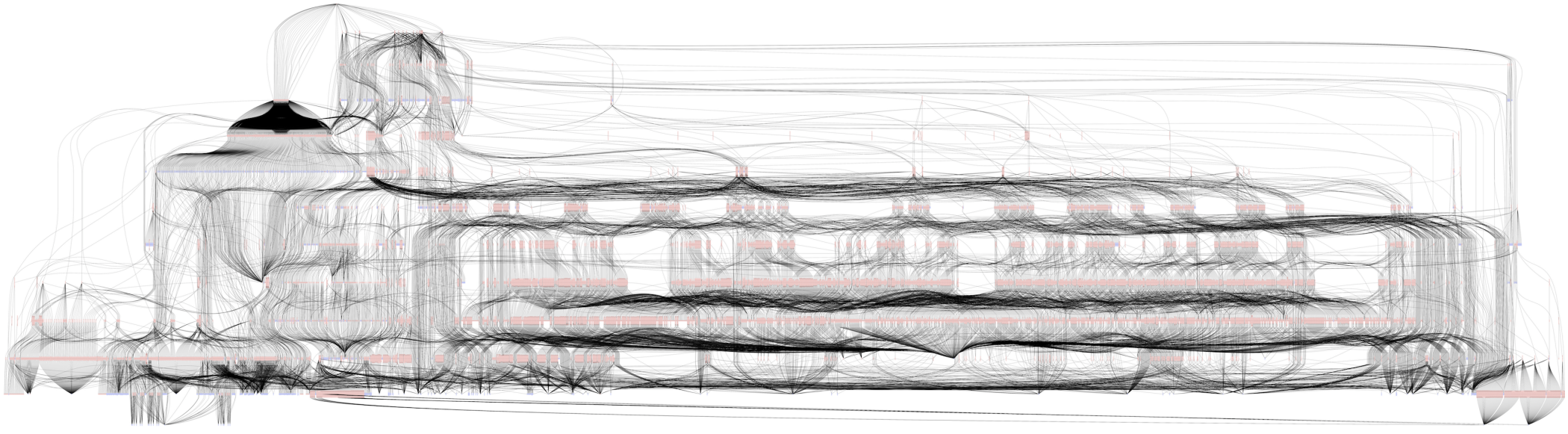
Exploit **full gradient of the likelihood** with **modern optimizers** to help speedup fit!

Gain this through the frameworks creating computational directed acyclic graphs and then applying the chain rule (to the operations)



# Tensor backends offer a computational advantage

For visual comparison: the computational graph of the Higgs discovery analysis from the C++ framework. Image courtesy of Kyle Cranmer.



# Publications using `pyhf`

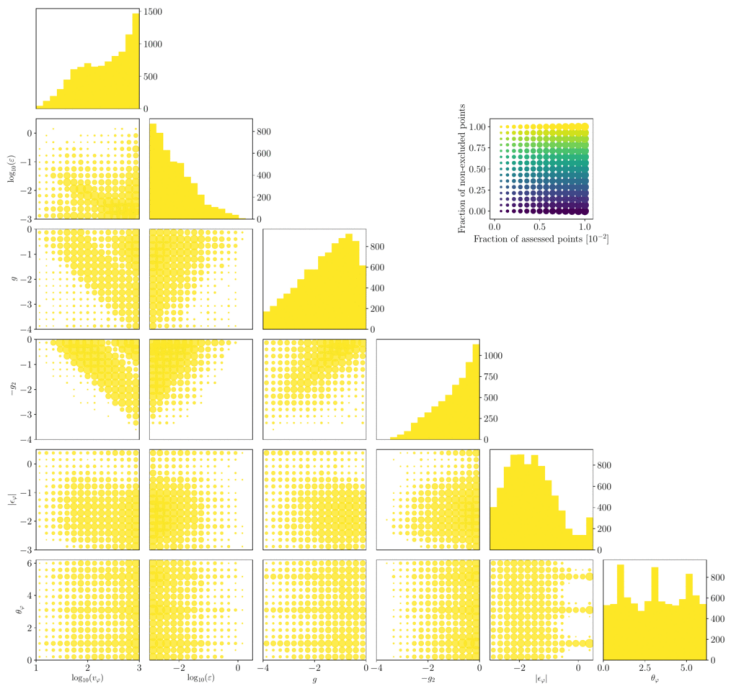
arXiv.org > hep-ph > arXiv:1810.05648

High Energy Physics - Phenomenology

**Constraining $A_4$ Leptonic Flavour Model Parameters at Colliders and Beyond**

Lukas Heinrich, Holger Schulz, Jessica Turner, Ye-Ling Zhou

*(Submitted on 12 Oct 2018)*




ATLAS Note

| | |
|---|---|
| Report number | ATL-PHYS-PUB-2019-029 |
| Title | **Reproducing searches for new physics with the ATLAS experiment through publication of full statistical likelihoods** |
| Corporate Author(s) | The ATLAS collaboration |

## New open release allows theorists to explore LHC data in a new way

**The ATLAS collaboration releases full analysis likelihoods, a first for an LHC experiment**

9 JANUARY, 2020 | By Katarina Anthony

Explore ATLAS open likelihoods on the HEPData platform (Image: CERN)

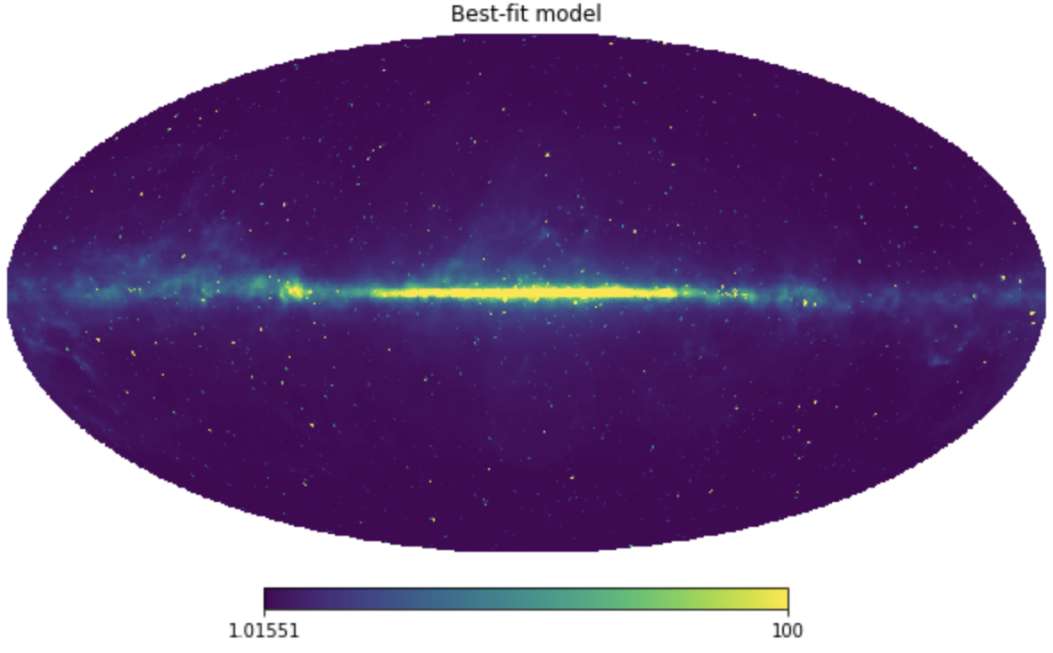
# Use in analysis outside of particle physics

- Public data from Fermi Large Area Telescope (LAT) analyzed by L. Heinrich et al.
- The LAT is a high-energy gamma-ray telescope — the gamma-ray photons come from extreme cosmological events
- Can represent the photons counts in the LAT as a binned model
  - Here full-sky map visualized with `healpy`'s Mollweide projection
  - Think: 2d histogram with special binning

In [6]:
```
m = pyhf.Model(spec, poiname = 'mu_dm')
bestfit = pyhf.optimizer.minimize(
    lambda theta,data,m: -m.logpdf(theta, data),data,m,
    init_pars = [1]*5,
    par_bounds = [[0,20]]*5
)
print(bestfit)
```

```
[ 0.89713789  0.43737808  1.0913045   0.75777142 13.3680226 ]
```

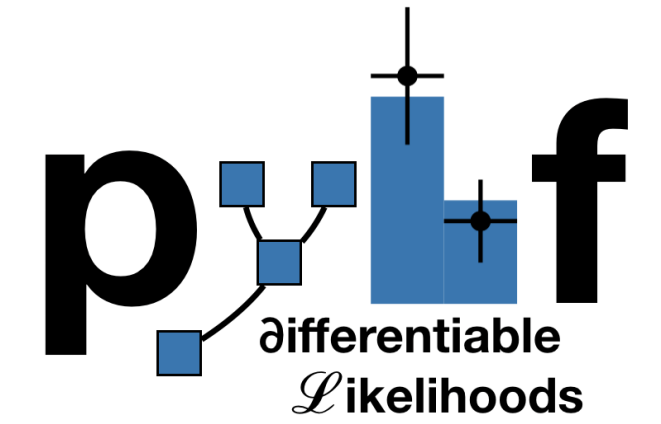
In [7]:
```
hp.mollview(m.expected_data(bestfit), max=100, title='Best-fit model', )
```



# Summary

`pyhf` provides:

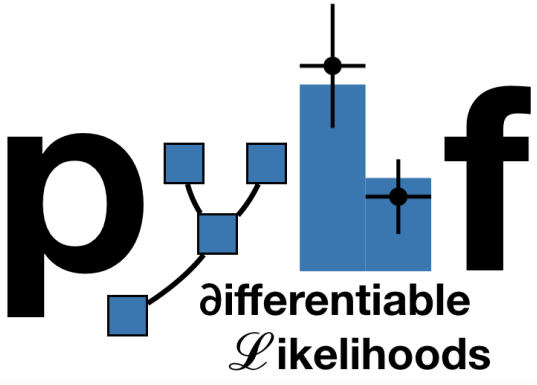
- **Accelerated** fitting library
  - reducing time to insight/inference!
  - Hardware acceleration on GPUs and vectorized operations
  - Backend agnostic Python API and CLI
- Flexible **declarative** schema
  - JSON: ubiquitous, universal support, versionable
- Enabling technology for **reinterpretation**
  - JSON Patch files for efficient computation of new signal models
  - Unifying tool for theoretical and experimental physicists
- Project in growing **Pythonic HEP ecosystem**
  - c.f. Jim Pivarski and Henry Schreiner's talks in High Performance Python track
  - Ask us about Scikit-HEP and IRIS-HEP!



# Thanks for listening!

# Come talk with us!

www.scikit-hep.org/pyhf


# Backup

# HistFactory Template (in more detail)

$$f\left(\vec{n},\vec{a}|\vec{\eta},\vec{\chi}\right) = \prod_{c\in\text{channels}}\prod_{b\in\text{bins}_c}\text{Pois}\left(n_{cb}|\nu_{cb}\left(\vec{\eta},\vec{\chi}\right)\right)\prod_{\chi\in\vec{\chi}}c_\chi\left(a_\chi|\chi\right)$$

$$\nu_{cb}(\vec{\eta},\vec{\chi}) = \sum_{s\in\text{samples}}\underbrace{\left(\sum_{\kappa\in\vec{\kappa}}\kappa_{scb}(\vec{\eta},\vec{\chi})\right)}_{\text{multiplicative}}\left(\nu^0_{scb}(\vec{\eta},\vec{\chi}) + \underbrace{\sum_{\Delta\in\vec{\Delta}}\Delta_{scb}(\vec{\eta},\vec{\chi})}_{\text{additive}}\right)$$

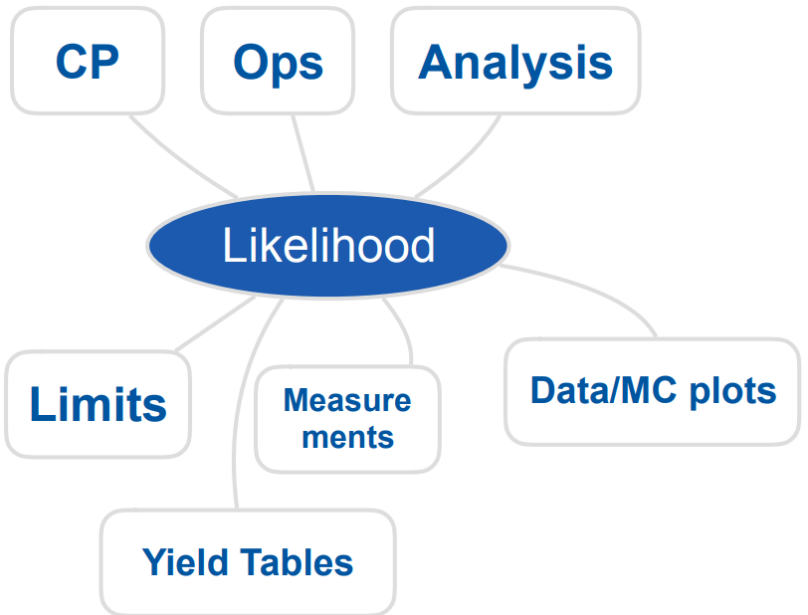
**Use:** Multiple disjoint channels (or regions) of binned distributions with multiple samples contributing to each with additional (possibly shared) systematics between sample estimates

**Main pieces:**

- Main Poisson p.d.f. for simultaneous measurement of multiple channels
- Event rates $\nu_{cb}$ from nominal rate $\nu^0_{scb}$ and rate modifiers $\kappa$ and $\Delta$
- Constraint p.d.f. (+ data) for "auxiliary measurements"
  - encoding systematic uncertainties (normalization, shape, etc)
- $\vec{n}$: events, $\vec{a}$: auxiliary data, $\vec{\eta}$: unconstrained pars, $\vec{\chi}$: constrained pars

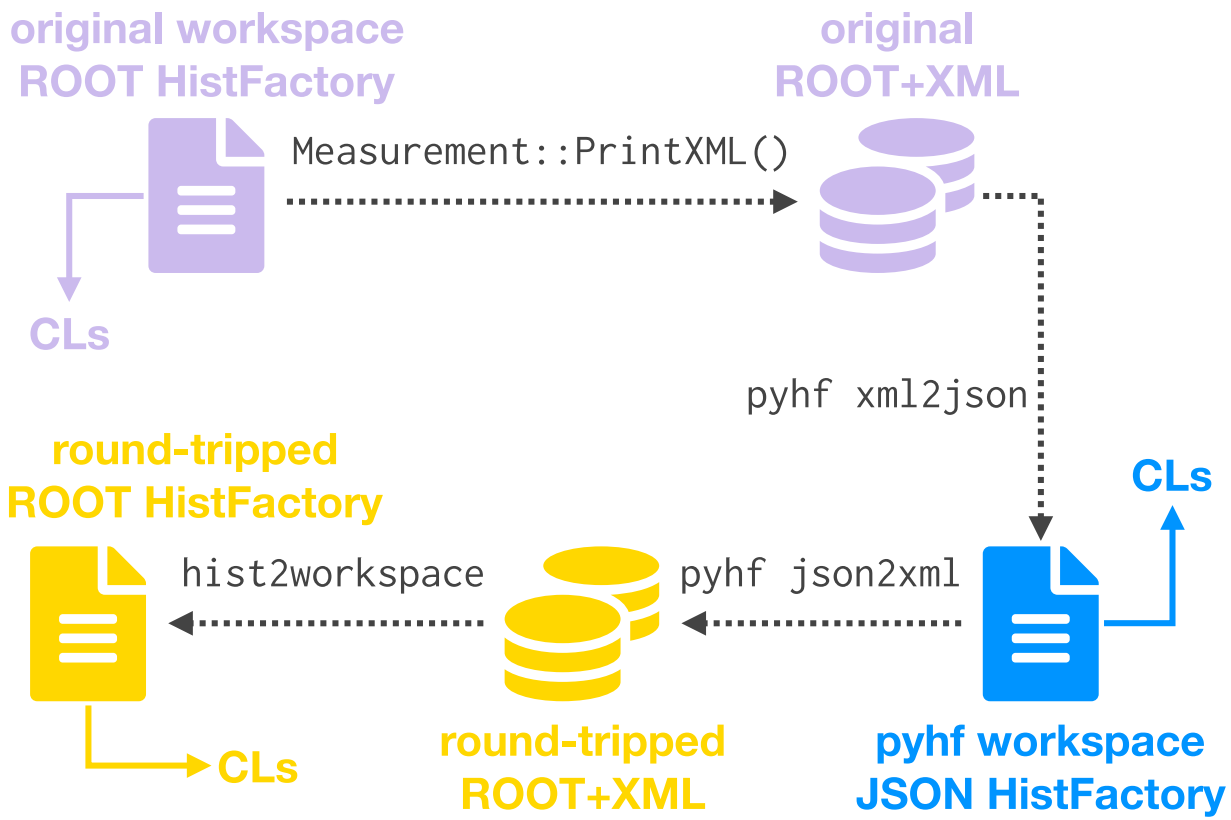
# Why is the likelihood important?

- High information-density summary of analysis
- Almost everything we do in the analysis ultimately affects the likelihood and is encapsulated in it
  - Trigger
  - Detector
  - Systematic Uncertainties
  - Event Selection
- Unique representation of the analysis to preserve



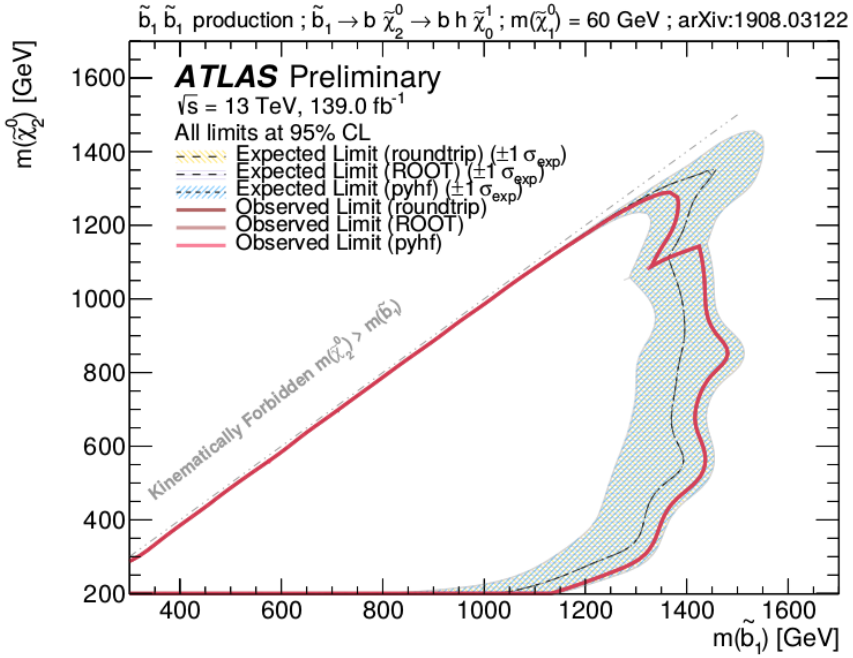
# Likelihood serialization and reproduction

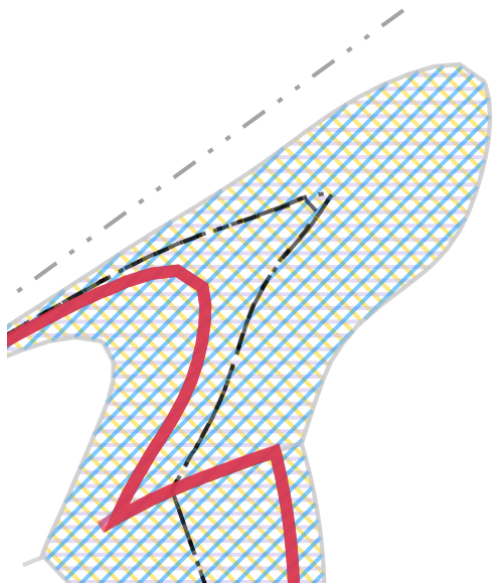
- ATLAS note on the JSON schema for serialization and reproduction of results [ATL-PHYS-PUB-2019-029]
  - Contours: original ROOT+XML, pyhf JSON, JSON converted back to ROOT+XML



# Likelihood serialization and reproduction

- ATLAS note on the JSON schema for serialization and reproduction of results [ATL-PHYS-PUB-2019-029]
  - Contours: original ROOT+XML, pyhf JSON, JSON converted back to ROOT+XML
    - Overlay of expected limit contours (hatching) and observed lines nice visualization of near perfect agreement
  - Serialized likelihood and reproduced results of ATLAS Run-2 search for bottom-squarks [JHEP12(2019)060] and published to HEPData
  - Shown to reproduce results but faster! **C++ (ROOT):** 10+ hours **pyhf:** < 30 minutes



# How are gradients computed when using NumPy?

As the NumPy backend with the SciPy optimizer doesn't support automatic differentiation, we make use of `scipy.optimize.minimize` along with the Sequential Least Squares Programming (SLSQP) method for the fit.

# How much of an affect does automatic differentiation have on fit speed?

This is hard to answer rigorously. It depends on the model complexity and what the analysis is. For a single fit for a small to medium model the use of gradients might not have much effect. However, for larger models the speedups from automatic differentiation become more apparent, but may not matter as much as JIT compilation.

In general though, lager more complex models derive greater benefits from automatic differentiation and JIT compilation.

# What SciPy optimizers are being used for MLE?

We use `scipy.optimize.minimize` along with the Sequential Least Squares Programming (SLSQP) method. We further leverage this through a custom `AutoDiffOptimizerMixin` class to feed the gradients from all the backends into `scipy.optimize.minimize` for performant optimization. In future versions (`v0.5.0` onwards), this mixin will be dropped in favor of tensor-backend shims.

# Is the NumPy backend competitive against C++?

Not for all models, but for some yes. If the model isn't too large and doesn't have a huge number of systematics, we can take advantage of the way the computations are laid out differently between `pyhf` and the C++ implementation and still be quite performant compared to the C++ thanks to vectorization.

# Can I use `pyhf` if I'm not a physicist?

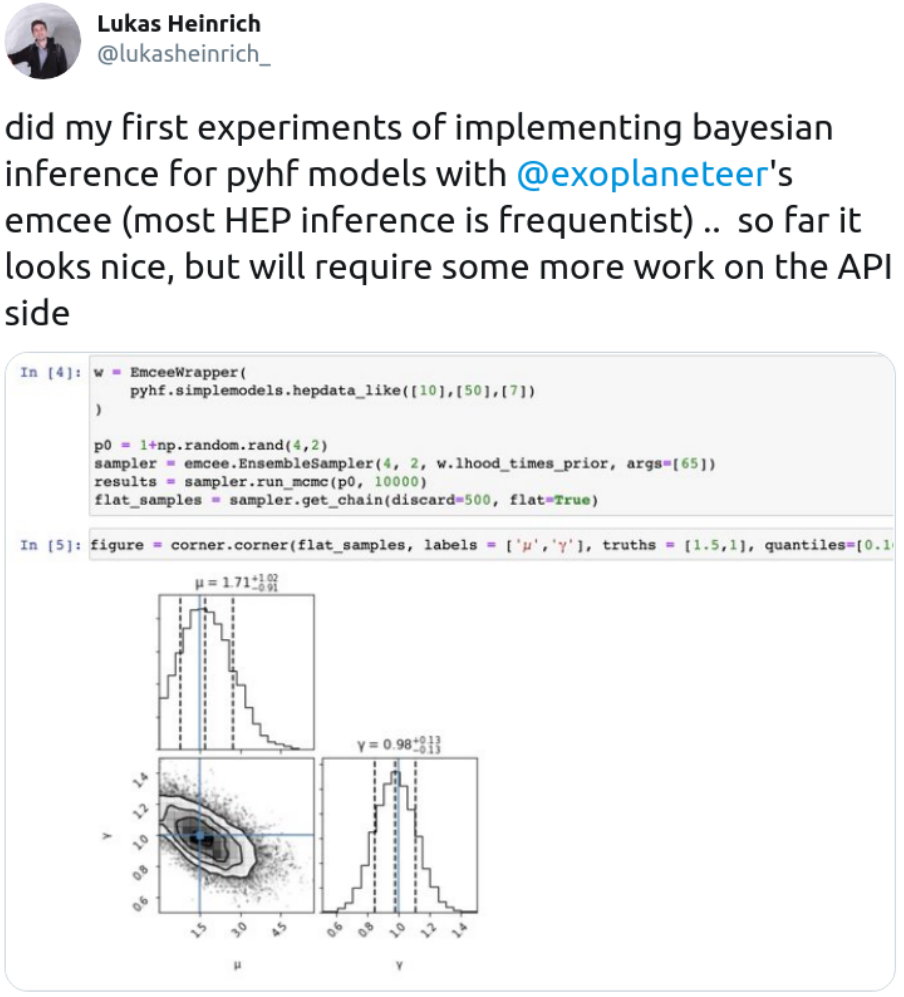
`pyhf` itself is focused exclusively on the HistFactory statistical model, but if you are performing counting experiments with template fits then maybe.

We hope that `pyhf` serves as an example of how models can be expressed in a declarative manner and implemented in a backend agnostic manner.

# Is this frequentist only?

Not exactly. The inference machinery that `pyhf` comes equipped with is frequentist focused, but the likelihood is common to both frequentist and Bayesian statistics. You could use the `pyhf` model in conjunction with priors to do Bayesian inference.

One of our "investigative" research areas is looking at using emcee and PyMC3 to do Bayesian inference with `pyhf` models.



# References


1. F. James, Y. Perrin, L. Lyons, *Workshop on confidence limits: Proceedings*, 2000.
2. ROOT collaboration, K. Cranmer, G. Lewis, L. Moneta, A. Shibata and W. Verkerke, *HistFactory: A tool for creating statistical models for use with RooFit and RooStats*, 2012.
3. L. Heinrich, H. Schulz, J. Turner and Y. Zhou, *Constraining $A_4$ Leptonic Flavour Model Parameters at Colliders and Beyond*, 2018.
4. A. Read, *Modified frequentist analysis of search results (the $\mathrm{CL}_s$ method)*, 2000.
5. K. Cranmer, *CERN Latin-American School of High-Energy Physics: Statistics for Particle Physicists*, 2013.
6. ATLAS collaboration, *Search for bottom-squark pair production with the ATLAS detector in final states containing Higgs bosons, b-jets and missing transverse momentum*, 2019
7. ATLAS collaboration, *Reproducing searches for new physics with the ATLAS experiment through publication of full statistical likelihoods*, 2019
8. ATLAS collaboration, *Search for bottom-squark pair production with the ATLAS detector in final states containing Higgs bosons, b-jets and missing transverse momentum: HEPData entry*, 2019

The end.