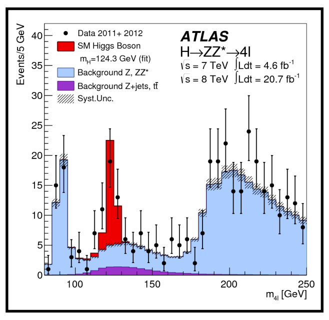
pyhf: auto-differentiable binned HEP likelihoods

Kyle Cranmer (NYU) Matthew Feickert (SMU)
Lukas Heinrich (CERN), Giordon Stark (UCSC)

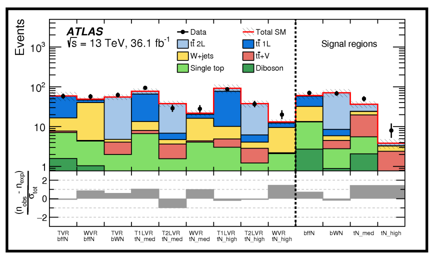
HistFactory

declarative binned likelihoods

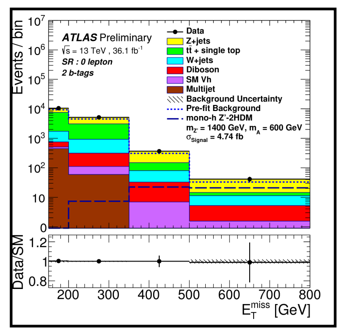
One of the most widely used statistical models in HEP for binned measuments and searches.



Standard Model



SUSY



Exotics

Mathematical Formulation:a parametrized p.d.f with parameters or interest (POI) and nuisance parameters

$$\mathcal{P}(n_c, x_e, a_p \,|\, \phi_p, \alpha_p, \gamma_b) = \boxed{\prod_{c\in\text{channels}} \left[\text{Pois}(n_c|\nu_c) \prod_{e=1}^{n_c} f_c(x_e|\alpha)\right]} \boxed{G(L_0|\lambda, \Delta_L) \cdot \prod_{p\in\mathbb{S}+\Gamma} f_p(a_p|\alpha_p)} \quad (1)$$

Primary Measurment:

- **multiple disjoint "channels" (e.g. event observables), each with multiple bins**
- **Poisson with rate parameters as functions of poi and nuisance parameters**

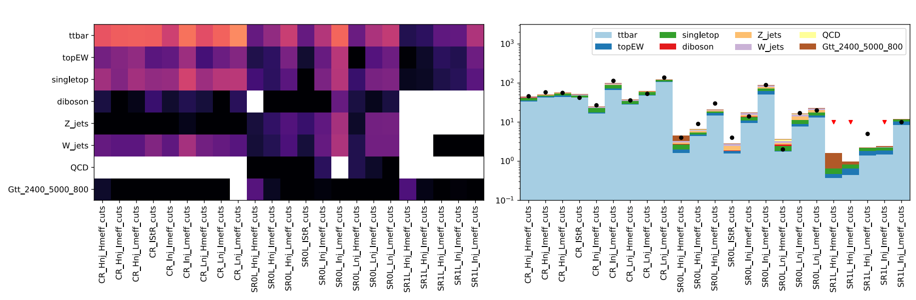
Auiliary Measurement

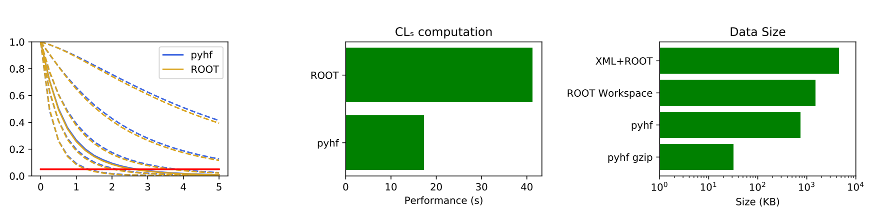
- **constraint terms on modeled as "measurements" of auxiliary data**

Performance

fast likelihood computation

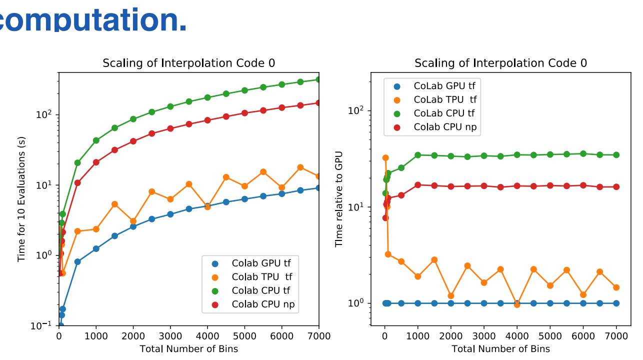
efficient use of tensor computation makes pyhf fast. Competitive with ROOT implementation - often faster.





Hardware Acceleration

For ML-library tensor backends Computational graph can be transparently placed on hardware accelerators: GPUs and TPUs for order of magnitude speed-up in computation.



Sharable Likelihoods

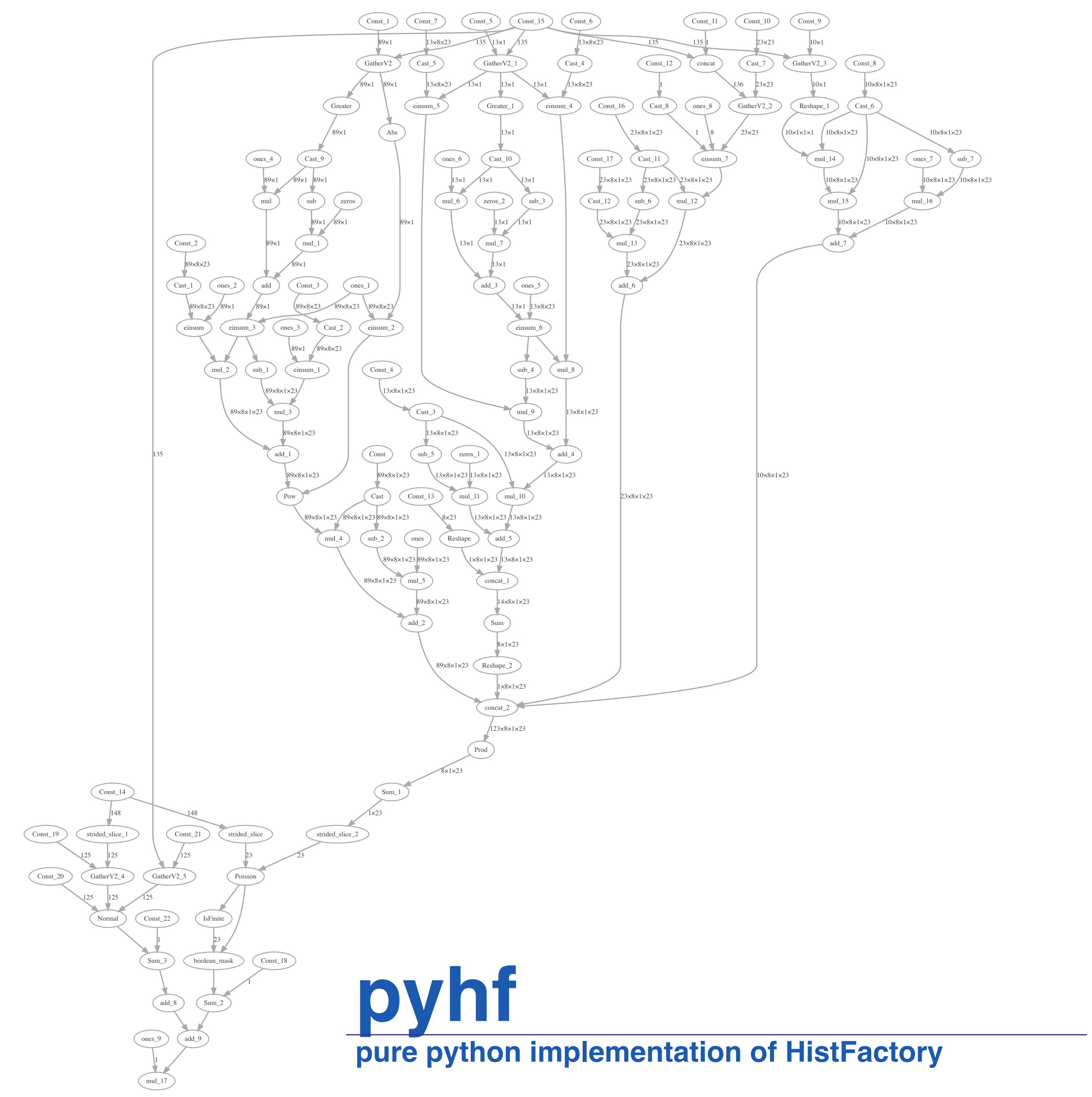
built for reinterpretation

For reinterpretation of searches, the bulk of the likelihood is invariant. Only need to patch the likelihood. Use JSONPatch standard to inject the new signal.

```
[
    {"op": "replace", "path": "/channels/0/samples/0/data", "value": [5., 6.]}
]
```

```
$> pyhf cls original.json|jq .CLs_obs
0.05290116065118097
```

```
$> pyhf cls original.json --patch newsignal.json|jq .CLs_obs
0.3401578753020146
```



pyhf

pure python implementation of HistFactory

implementation of HistFactory likelihood (1) as a computational graph of multi-dimensional array operations.

Use of array ("tensor") operations through a common API layer around high-performance tensor libraries: e.g.

NumPy TensorFlow PyTorch

Installation:

```
$> pip install pyhf
```

Example: simple number-counting experiment

```
import pyhf
pdf = pyhf.simplemodels.hepdata_like(
  signal_data=[12.0, 11.0], bkg_data=[50.0, 52.0], bkg_uncerts=[3.0, 7.0]
)
CLs_obs, CLs_exp = pyhf.utils.hypotest(1.0, [51, 48] + pdf.config.auxdata, pdf, return_expected=True)


print('Observed: {}, Expected: {}'.format(CLs_obs, CLs_exp))
Observed: [0.05290116], Expected: [0.06445521]
```

Auto-Differentiation:

Tensor libraries from ML communty provide exact gradients for use in minimization.

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

Optimizers

pyhf likeliehood are simple tensor-value python functions. Can use multiple minimization algorithms, such as `scipy.minimize` or MINUIT

JSON Format

The full likelihood can be expressed as a single JSON document

- **easy archivability (HepData)**
- **easy sharing across network**
- **easy manipulation**

```
$> curl http://url-to-json/workspace.json|pyhf cls
{
    "CLs_exp": [
        0.002606408505279359,
        0.013820656047622592,
        0.0644552079856191,
        0.23526102499555396,
        0.573041803728844
    ],
    "CLs_obs": 0.05290116065118097
}
```

```
toplvl:
  measurements:
  - {name: demo, config: {poi: mu}}
data:
  singlechannel: [51, 48]
channels:
- name: singlechannel
  samples:
  - name: signal
    data: [12, 11]
    modifiers:
    - name: mu
      type: normfactor
      data: null
  - name: background
    data: [50,52]
    modifiers:
    - name: uncorr_bkguncrt
      type: shapesys
      data: [3,7]
```