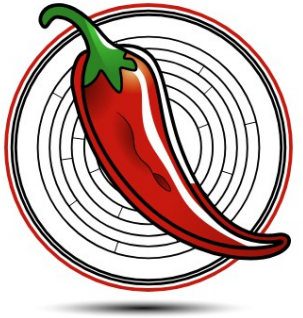
Pepper: a CMS columnar analysis framework

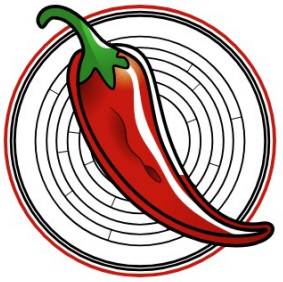
Mads H. Baatrup, Laurids Jeppe, Dominic Stafford

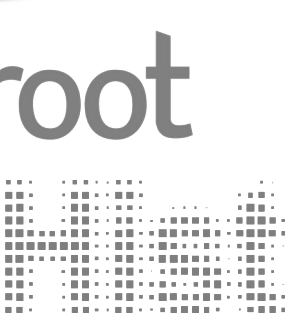


07.09.2026
PyHEP Dev Workshop

What is Pepper?

- A coffea-based, CMS approved, analysis framework
- Last step, building on entire sci-kit HEP ecosystem
- Gives concrete implementations for:
  - Object selections and corrections
  - Scale-out
  - Histogram book-keeping/plotting
- Lots of direct contact between maintainers and users





Pepper philosophy

- Analyses start from CMS NanoAOD format
  - Lightweight, flat ntuples
- Initially developed by two PhD students on DESY cluster
  - Influenced decisions around scale out (HTCondor), and data access (can have all files on-site)
  - Now spread to further sites, e.g. CERN lxplus and SPRACE (Brazil)
- Aim to produce histograms (or per-event inputs for ML) in one step to have fast time-to-insight

Code structure

- Users write their analysis in a Processor class:
  - Inherited from coffea
  - Implementations of cuts and SFs for CMS standard objects
- Book-keeping handled by Selector class:
  - Keeps track of cuts, SFs and systematics
  - Calls processor functions which define these
- Automatically fills histograms after each step (configurable)

Main process function

```
selector.add_cut("NoAddLeps",
                 partial(self.no_additional_leptons, is_mc))
selector.set_column("Electron", self.pick_electrons)
selector.set_column("Muon", self.pick_muons)
selector.set_column("Lepton", partial(
    self.build_lepton_column, is_mc, selector.rng))
# Wait with hists filling after channel masks are available
selector.add_cut("AtLeast2Leps", partial(self.lepton_pair, is_mc),
                 no_callback=True)
selector.set_multiple_columns(self.channel_masks)
selector.set_cat("channel", {"is_ee", "is_em", "is_mm"})
selector.set_column("mll", self.mass_lepton_pair)
selector.set_column("dilep_pt", self.dilep_pt, lazy=True)
```

Apply lepton object cuts (config controlled)

Combine into one column

Apply two lepton selection

Split into categories on lepton flavour

Set columns based on dilepton kinematics

Gitlab link

Cut function example

```
def lepton_pair(self, is_mc, data):
    """Select events that contain at least two leptons."""
    accept = np.asarray(ak.num(data["Lepton"]) >= 2)
    if is_mc:
        weight, systematics = self.compute_lepton_sf(data[accept])
        accept = accept.astype(float)
        accept[accept.astype(bool)] *= np.asarray(weight)
        return accept, systematics
    else:
        return accept
```

Basic selection

Compute SFs and systematics for MC

Combine SF weights with selection

Gitlab link

Configuration

```
"ele_cut_transreg": true,
"ele_eta_min": -2.4,
"ele_eta_max": 2.4,
"good_ele_id": "mva:Iso90",
"good_ele_pt_min": 20.0,
"additional_ele_id": "mva:Iso90",
"additional_ele_pt_min": 20.0,

"electron_sf": [
    [
        "/cvmfs/cms-griddata.cern.ch/cat/metadata/EGM/Run3-24CDEReprocessingFGHIPrompt-Summer24-NanoAODv15/2025-12-15/electron.json.gz",
        "Electron-ID-SF",
        {"year": "2024Prompt", "WorkingPoint": "wp80iso"}
    ],
    [
        "/cvmfs/cms-griddata.cern.ch/cat/metadata/EGM/Run3-24CDEReprocessingFGHIPrompt-Summer24-NanoAODv15/2025-12-15/electron.json.gz",
        "Electron-ID-SF",
        {
            "year": "2024Prompt",
            "WorkingPoint": {
                "RecoBelow20": [0, 20],
                "Reco20to75": [20, 75],
                "RecoAbove75": [75, "inf"]
            }
        }
    ]
],
```

Electron object definition

Specify SFs from CMS correctionlib json files

Gitlab link

Histogramming

```
"leading_electron_pt": {
    "bins": [
        {
            "name": "pt",
            "label": "Electron $p_{\\mathrm{T}}$",
            "n_or_arr": 100,
            "lo": 0,
            "hi": 400,
            "unit": "GeV"
        }
    ],
    "fill": {
        "pt": [
            "Electron",
            "pt",
            {"leading": 1}
        ]
    }
},
```

Gitlab link

- By default histograms produced after each cut
  - Can restrict cuts, etc. to save memory
- Histograms defined in configuration
  - Can also be multi-dimensional
- Can save in root or hist (coffea pickle) format
  - We provide an extensible plotting script

User experience

- Used in ~10 published and ~15 in progress analyses
  - Mostly at DESY, but gradually expanding to other sites
- Aligns well with python skills of most new students
- Generally can run an analysis (1 year, NanoAOD to plots) in ~1 hour
  - Depending on computing cluster
- Framework developers get large range of questions:
  - Technical issues
  - “How do I do X in a columnar way”
  - Physics/analysis strategy questions

Performance bottlenecks

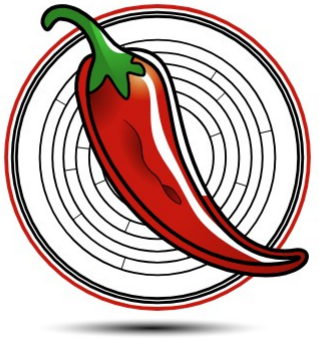
- Most common bottle neck is memory, can reach O(10GB) per worker
  - Slows time to get jobs
  - Doesn’t align with Cluster assumptions about job requirements
  - Users don’t always realise the issue or the cause
- Culprit 1: Large data columns
  - Could be fixed by column unloading
  - Not trivial, e.g. full copies of columns may be stored in derived quantities
- Culprit 2: Histograms
  - Could be alleviated by histserv; aim to try this week

Coding challenges

- Some problems, particularly combinatorics, more challenging in columnar frameworks
  - Various awkward helper functions exist, but not always clear which to use
- 4-vector manipulation can pose challenges:
  - Types of derived classes often change
  - Setting energy of 4-vector in p_T, η, ϕ, mass basis will not have desired effect
- Internal awkward structures can become confusing
  - Pepper has 40 lines of code to handle edge cases when saving awkward arrays to ROOT
- Still work to do on making these features more intuitive
- Or better educating users

Summary

- Pepper is a full CMS analysis framework
  - Already used to publish ~10 analyses
  - Easily adopted by many new students
- Wouldn’t be possible without scikit-hep ecosystem
- Still some common pitfalls:
  - Memory bottlenecks
  - Columnar coding challenges
- Development always necessary to keep up with analysis complexity

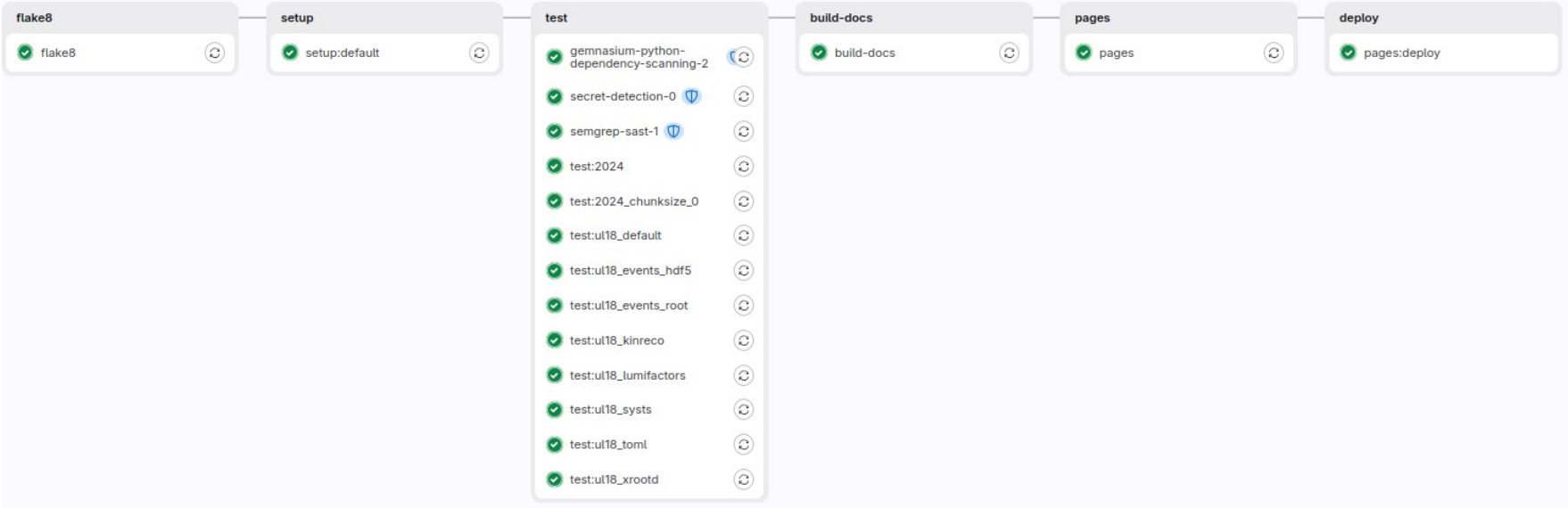


Gitlab link

Backup

Documentation and CI

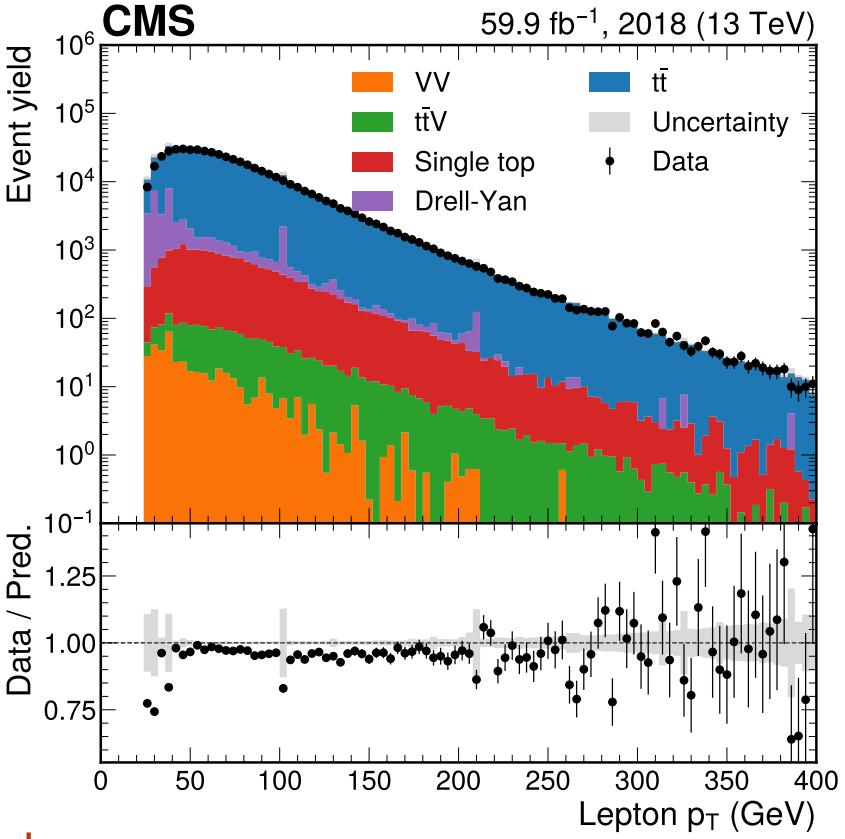
- Have CI pipelines for code style (flake8) and consistency checks
- One pipeline builds the docs
- Also implement CMS analysis recipes (consistency checks CMS object corrections are applied as recomended)



Plotting

- Configurable plotting script for hist histograms
- Intended as starting point for users

```
"backgrounds": {
    "DY": {
        "label": "Drell-Yan",
        "color": "tab:purple",
        "datasets": [
            "DYJetsToLL_M-10to50_TuneCP5_13TeV-madgraphMLM-pythia8",
            "DYJetsToLL_M-50_TuneCP5_13TeV-madgraphMLM-pythia8"
        ]
    },
    "TT": {
        "label": "$t \\bar{t}$",
        "color": "tab:blue",
        "datasets": [
            "TTTo2L2Nu_TuneCP5_13TeV-powheg-pythia8",
            "TTToSemiLeptonic_TuneCP5_13TeV-powheg-pythia8"
```



Gitlab link

Awkward edge-case handling

```
if pepper.misc.akismasked(array):
    is_none = ak.is_none(array)
    # sometimes there are no actual Nones even though the type says that
    # there should be, because of an issue in ak.to_packed:
    # https://github.com/scikit-hep/awkward/issues/3939
    # workaround by checking explicitly and dropping nones
    if not ak.any(is_none):
        array = ak.drop_none(array)
    else:
        if "mask" + key in arrays:
            raise RuntimeError(f"Output named 'mask{key}' already present "
                               "but need this key for storing the mask")
        ret["mask" + key] = ~is_none
        if array.ndim > 1:
            # 2D array with masked values in the first axis
            # will cause trouble for uproot - replace Nones by empty lists
            array = ak.fill_none(array, [], axis=0)
        elif len(array.fields) > 0:
            array = ak.fill_none(array, {k: 0 for k in array.fields})
        else:
            array = ak.fill_none(array, 0)

# Check for case mask is at level of individual fields
if any([pepper.misc.akismasked(array[k]) for k in array.fields]):
    counts = None
    can_zip = True
    for k in array.fields:
        if "mask" + key not in arrays:
            ret["mask" + key] = ~ak.is_none(array[k])
        elif ret["mask" + key] != ~ak.is_none(array[k]):
            raise RuntimeError(
                f"Inconsistent masks for fields of key {key}. Please "
                f"output these as separate columns")
        if (can_zip and array[k].ndim > 1):
            if counts is None:
                counts = ak.num(array[k])
            else:
                can_zip = ak.all(counts == ak.num(array[k]))
        else:
            can_zip = False
    if can_zip:
        array = ak.zip({k: ak.fill_none(array[k], 0) for k in array.fields})
    else:
        array = ak.Array({k: ak.fill_none(array[k], 0) for k in array.fields})
```