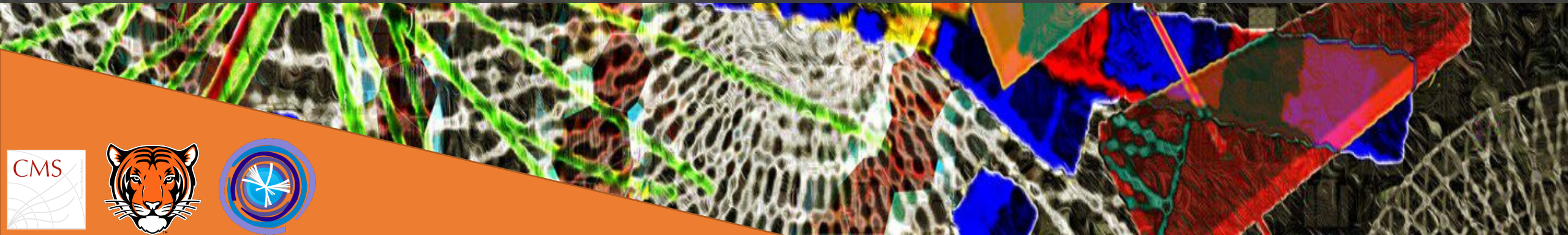


Implementation of systematics in AGC with Coffea 2025

Massimiliano Galli

IRIS-HEP/AGC Demo Day 7

14 February 2025



About me

- Postdoc @ Princeton University since mid-January
- Main interests/areas of work:
 - So far: uproot (internals investigation and bugfixing), AGC
 - Moving forward: statistical tools, application of SBI techniques to CMS analyses
- Previously:
 - PhD @ ETH Zürich
 - Full Run 2 Higgs diff. XS combination with ETF-kappa interpretation
 - Development of tools and analysis techniques for Hgg Run 3 analysis
 - CERN Tech Student in the ROOT team
 - PyROOT with Cppyy
 - Multiple-python build system



Task

- The Analysis Grand Challenge ([link](#)) provides examples of end-to-end analysis with modern pythonic packages, at scale
- [Ongoing demo](#) showcasing ttbar-like analysis with CMS open data with coffea2025
- Task:
 - Implement corrections and systematics in the migration demo (keeping a user-oriented mindset, i.e. trying to spot possible challenges/tricky parts for users)
 - Perform basic performance tests to assess the improvements provided by coffea2025 and dask-awkward



Systematics

- Object-level:
 - Jet energy scale
 - Jet energy resolution
- Event-level:
 - Signal modelling (scale variations, matrix element computation, parton shower)
 - B-tagging variations
 - Scale variations for W +jets

Systematics

- Object-level:
 - Jet energy scale
 - Jet energy resolution
- Event-level:

Simple multiplication,
no changes in API

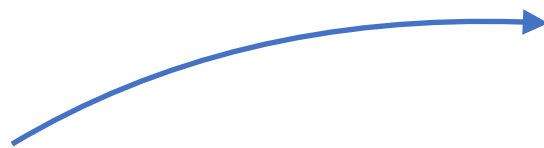
Different
samples, no
changes in API

- Signal modelling (scale variations, matrix element computation, parton shower)
- B-tagging variations
- Scale variations for W +jets

No changes in
API thanks to
adaptations in
correctionlib

Systematics

- Object-level:
 - Jet energy scale
 - Jet energy resolution
- Event-level:
 - Signal modelling (scale variations, matrix element computation, parton shower)
 - B-tagging variations
 - Scale variations for W+jets



Change in API
required

Adapting systematics

```
events["pt_res_up"] = utils.systematics.jet_pt_resolution(events.Jet.pt)
```

```
import awkward as ak
import numpy as np

# functions creating systematic variations
def jet_pt_resolution(pt):
    # normal distribution with 5% variations, shape matches jets
    counts = ak.num(pt)
    pt_flat = ak.flatten(pt)
    resolution_variation = np.random.normal(np.ones_like(pt_flat), 0.05)
    return ak.unflatten(resolution_variation, counts)
```

```
File "/work/gallim/mambaforge/envs/250131_AGC/lib/python3.13/site-packages/dask_awkward/lib/core.py", line 1718, in __array__
    raise NotImplementedError
NotImplementedError
```



Adapting systematics

```
events["pt_res_up"] = dak.map_partitions(rand_gauss, events.Jet.pt)
```

```
def rand_gauss(item):  
    # from https://github.com/scikit-  
    hep/coffea/blob/master/src/coffea/jetmet_tools/CorrectedJetsFactory.py#L42  
    seeds = (  
        ak.flatten(ak.typtracer.length_one_if_typtracer(item)).to_numpy().view("i4")  
    )  
    randomstate = np.random.Generator(np.random.PCG64(seeds))  
  
    def getfunction(layout, depth, **kwargs):  
        if isinstance(layout, ak.contents.NumpyArray) or not isinstance(  
            layout, (ak.contents.Content,))  
        ):  
            return ak.contents.NumpyArray(  
                randomstate.normal(loc=1, scale=0.05, size=len(layout)).astype(np.float32)  
            )  
        return None  
  
    out = ak.transform(  
        getfunction,  
        ak.typtracer.length_zero_if_typtracer(item),  
        behavior=item.behavior,  
    )  
    if ak.backend(item) == "typtracer":  
        out = ak.Array(  
            out.layout.to_typtracer(forget_length=True), behavior=out.behavior  
        )  
  
    assert out is not None  
    return out
```

Use `map_partitions` to generate a smearing for each partition

- Sets a deterministic seed based off the data and sample from it (within the function in each partition)
- Properly handle typtracer and conversion from numpy to dask-awkward

Suggested by Lindsey, from [here](#)

Comparison with old Coffea

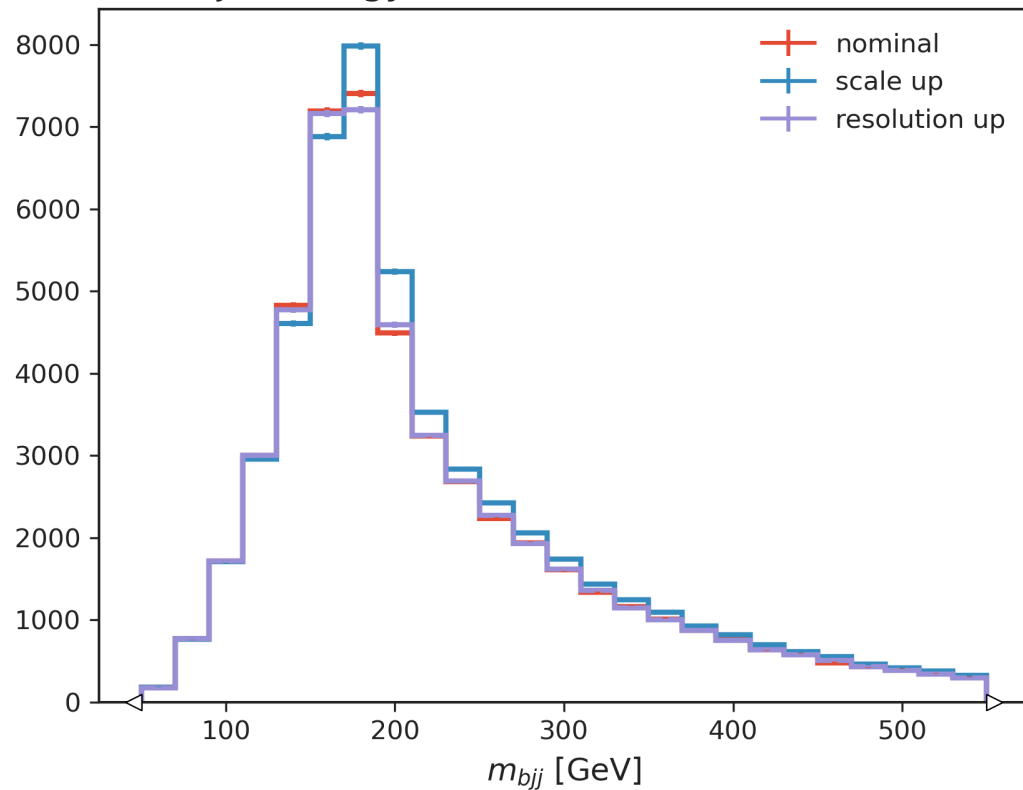
- Run basic performance tests to compare with old coffea
- Setup:
 - Comparison scripts: [old](#) vs [new](#)
 - Use a Dask LocalCluster to avoid time taken to spawn the jobs to affect the results
 - Check compute time as a function of number of files per sample, chunksize, number of workers
 - In all configurations, use the 9 datasets in the prototype
 - For new coffea, include both the time taken to create the task graph and to compute

Histogram Comparison

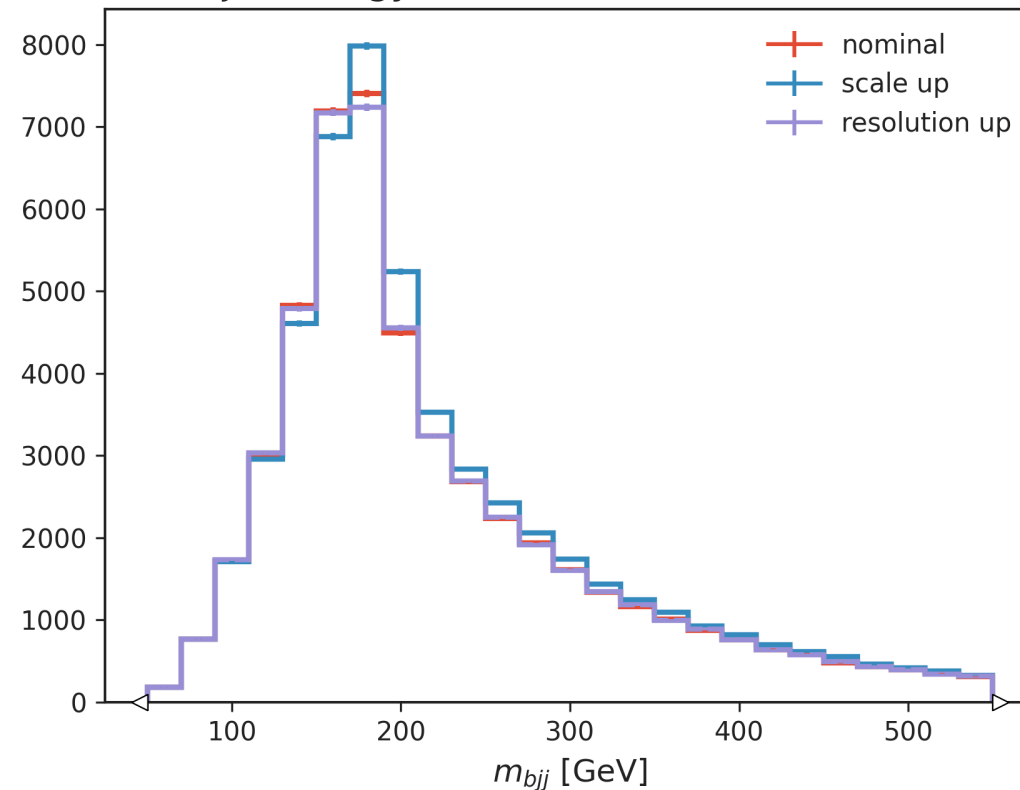
New

Old

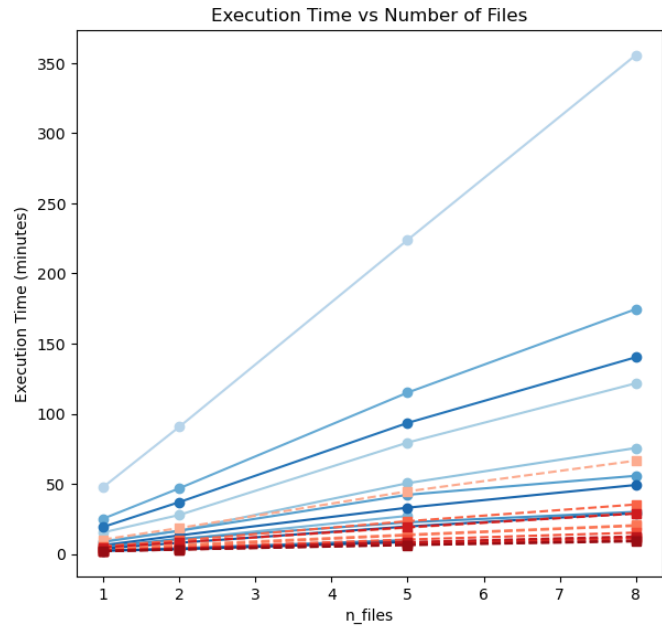
jet energy scale/resolution variations



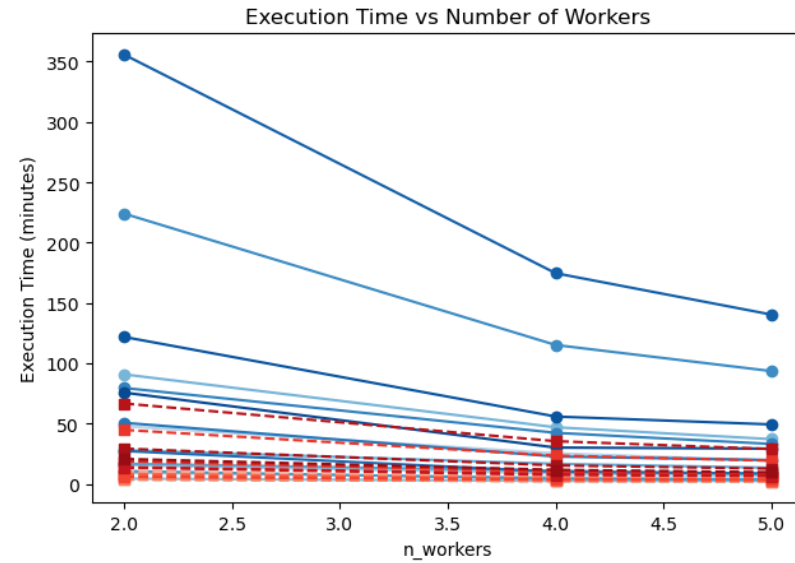
jet energy scale/resolution variations



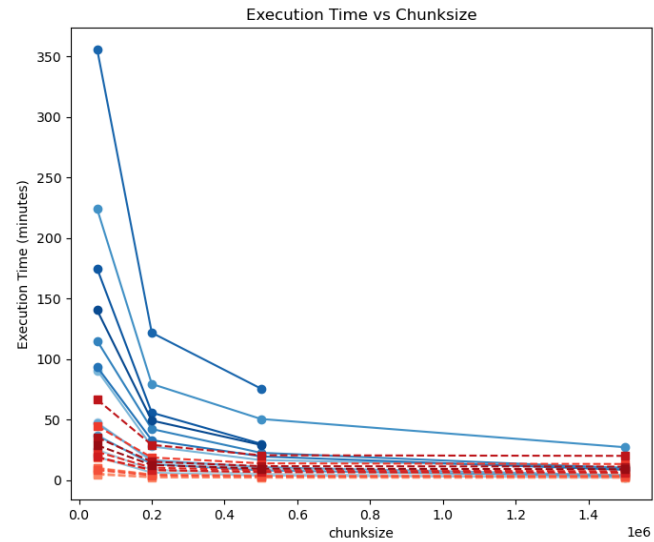
Performance Comparisons



- Old - n_workers=2, chunksize=50000
- Old - n_workers=2, chunksize=200000
- Old - n_workers=2, chunksize=500000
- Old - n_workers=2, chunksize=1500000
- Old - n_workers=4, chunksize=50000
- Old - n_workers=4, chunksize=200000
- Old - n_workers=4, chunksize=500000
- Old - n_workers=4, chunksize=1500000
- Old - n_workers=5, chunksize=50000
- Old - n_workers=5, chunksize=200000
- Old - n_workers=5, chunksize=500000
- Old - n_workers=5, chunksize=1500000
- New - n_workers=2, chunksize=50000
- New - n_workers=2, chunksize=200000
- New - n_workers=2, chunksize=500000
- New - n_workers=2, chunksize=1500000
- New - n_workers=4, chunksize=50000
- New - n_workers=4, chunksize=200000
- New - n_workers=4, chunksize=500000
- New - n_workers=4, chunksize=1500000
- New - n_workers=5, chunksize=50000
- New - n_workers=5, chunksize=200000
- New - n_workers=5, chunksize=500000
- New - n_workers=5, chunksize=1500000



- Old - n_files=1, chunksize=50000
- Old - n_files=1, chunksize=200000
- Old - n_files=1, chunksize=500000
- Old - n_files=1, chunksize=1500000
- Old - n_files=2, chunksize=50000
- Old - n_files=2, chunksize=200000
- Old - n_files=2, chunksize=500000
- Old - n_files=2, chunksize=1500000
- Old - n_files=5, chunksize=50000
- Old - n_files=5, chunksize=200000
- Old - n_files=5, chunksize=500000
- Old - n_files=5, chunksize=1500000
- Old - n_files=8, chunksize=50000
- Old - n_files=8, chunksize=200000
- Old - n_files=8, chunksize=500000
- Old - n_files=8, chunksize=1500000
- New - n_files=1, chunksize=50000
- New - n_files=1, chunksize=200000
- New - n_files=1, chunksize=500000
- New - n_files=1, chunksize=1500000
- New - n_files=2, chunksize=50000
- New - n_files=2, chunksize=200000
- New - n_files=2, chunksize=500000
- New - n_files=2, chunksize=1500000
- New - n_files=5, chunksize=50000
- New - n_files=5, chunksize=200000
- New - n_files=5, chunksize=500000
- New - n_files=5, chunksize=1500000
- New - n_files=8, chunksize=50000
- New - n_files=8, chunksize=200000
- New - n_files=8, chunksize=500000
- New - n_files=8, chunksize=1500000



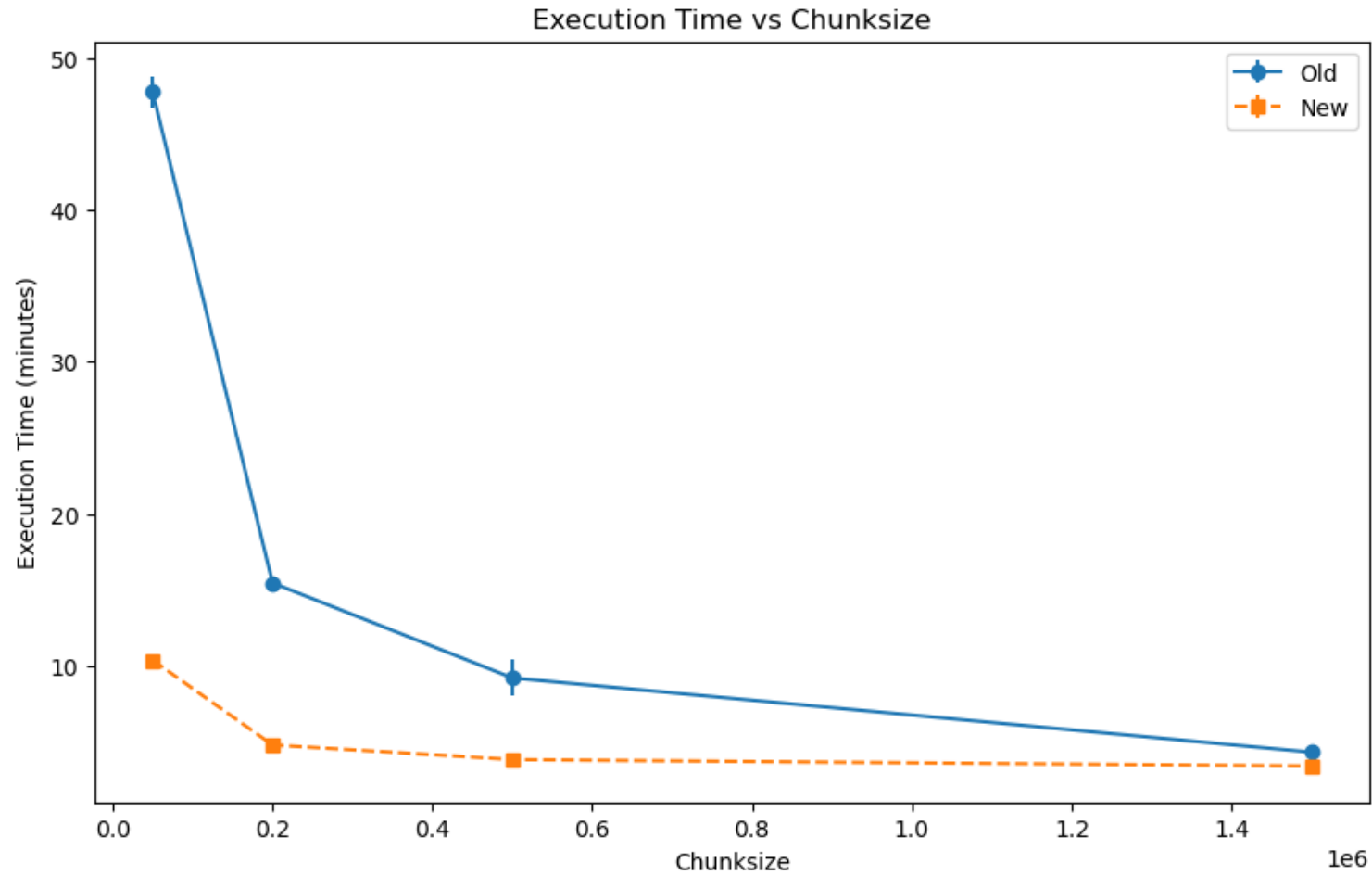
- Old - n_files=1, n_workers=2
- Old - n_files=1, n_workers=4
- Old - n_files=1, n_workers=5
- Old - n_files=2, n_workers=2
- Old - n_files=2, n_workers=4
- Old - n_files=2, n_workers=5
- Old - n_files=5, n_workers=2
- Old - n_files=5, n_workers=4
- Old - n_files=5, n_workers=5
- Old - n_files=8, n_workers=2
- Old - n_files=8, n_workers=4
- Old - n_files=8, n_workers=5
- New - n_files=1, n_workers=2
- New - n_files=1, n_workers=4
- New - n_files=1, n_workers=5
- New - n_files=2, n_workers=2
- New - n_files=2, n_workers=4
- New - n_files=2, n_workers=5
- New - n_files=5, n_workers=2
- New - n_files=5, n_workers=4
- New - n_files=5, n_workers=5
- New - n_files=8, n_workers=2
- New - n_files=8, n_workers=4
- New - n_files=8, n_workers=5

* for new coffea, values also include the time to build the task graphs (~34 seconds)



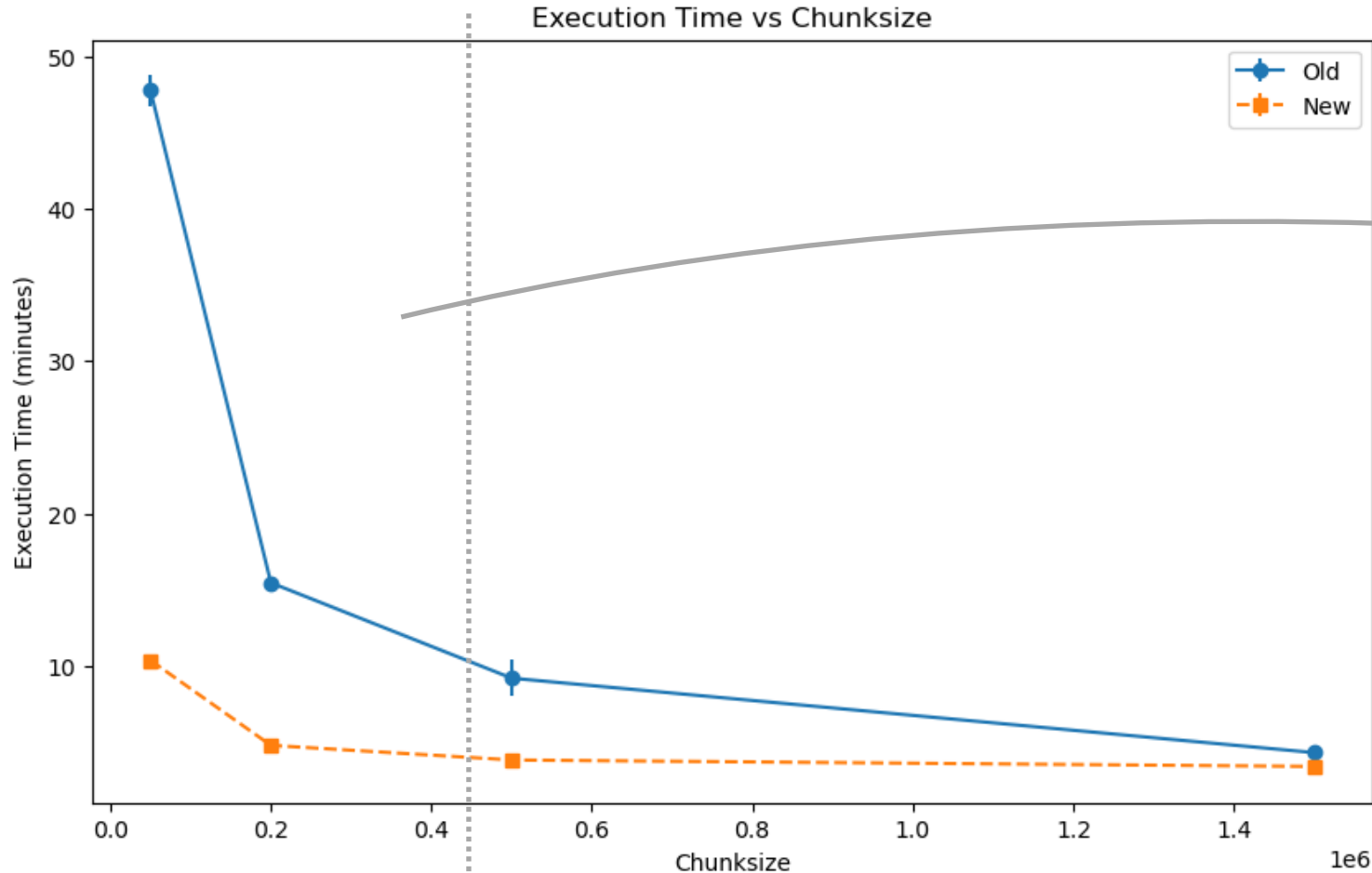
Performance Comparisons

1 file per sample, 2 workers



Performance Comparisons

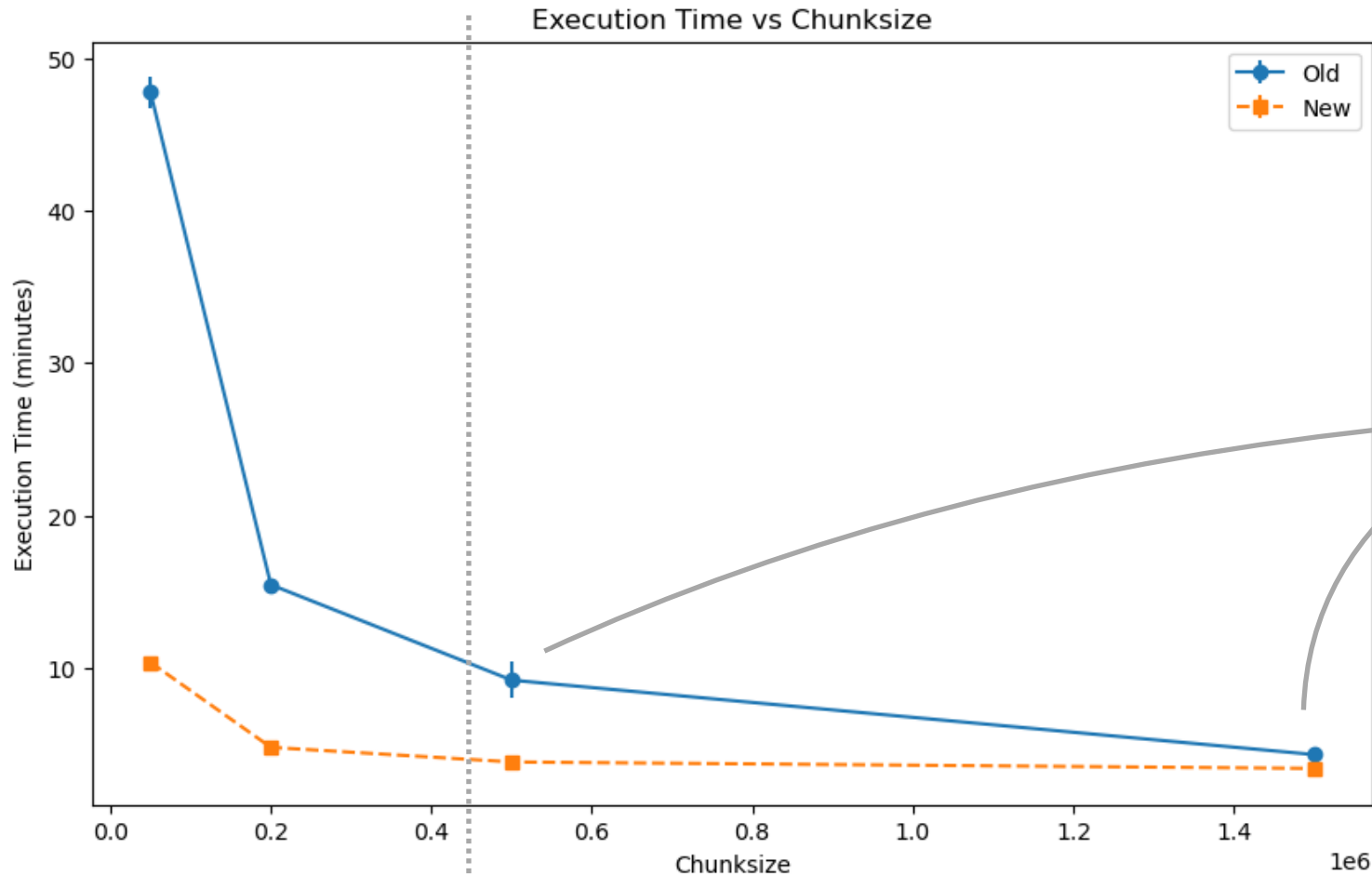
1 file per sample, 2 workers



- Execution time includes both python overhead and “number crunching”
- Small chunksize -> more partitions -> more python overhead
- Two PRs decreased python overhead in awkward [#3359](#) and vector [#554](#)
- The effect of the overhead reduction is thus particularly visible when the chunksize is lower

Performance Comparisons

1 file per sample, 2 workers

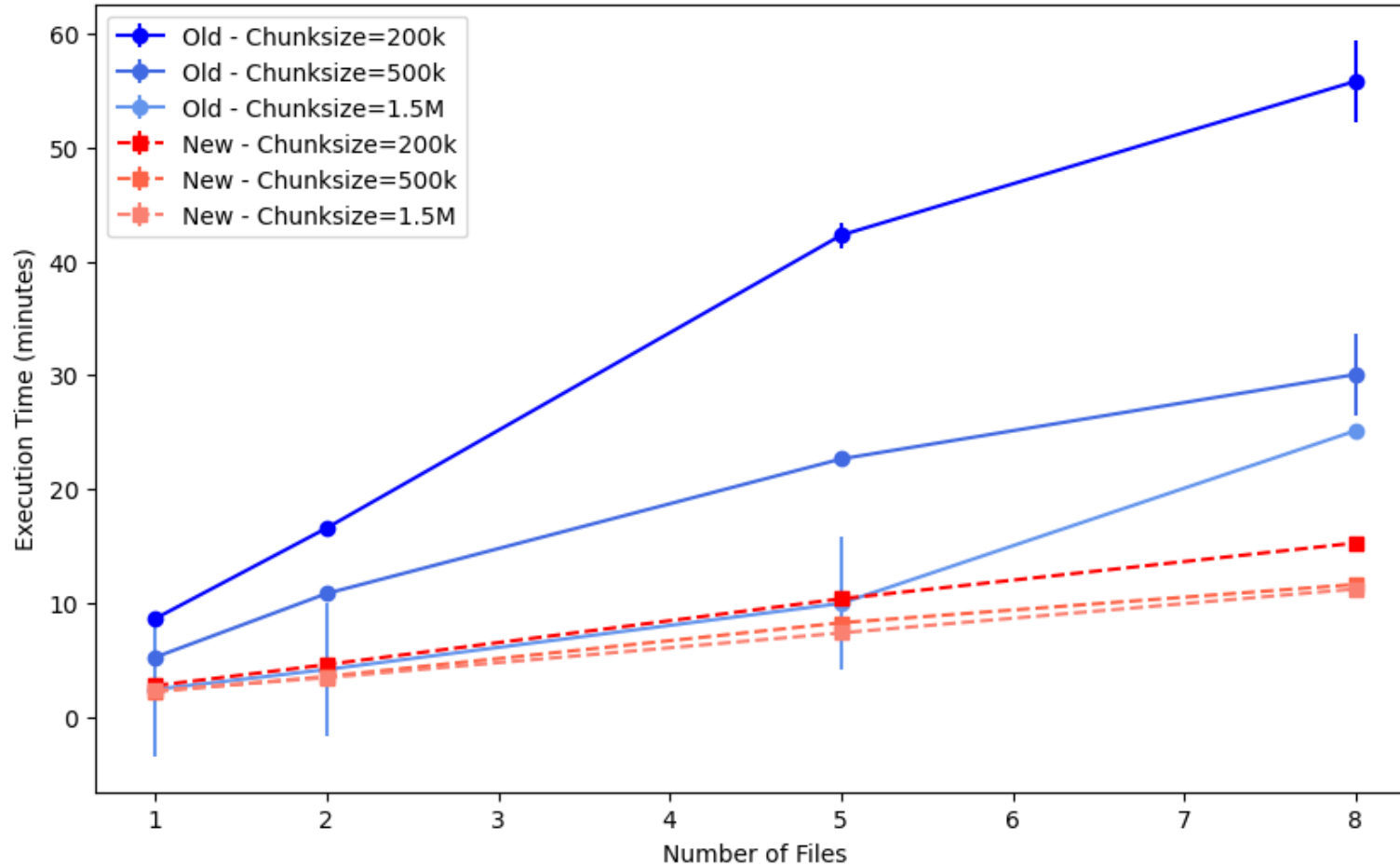


- The effect becomes less noticeable when the chunksize increases (we have less partitions)
- In the extreme case of only one partition, the python overhead component becomes negligible and we have similar execution times (no improvements in the “number crunching” part)

Performance Comparisons

4 workers

Execution Time vs Number of Files

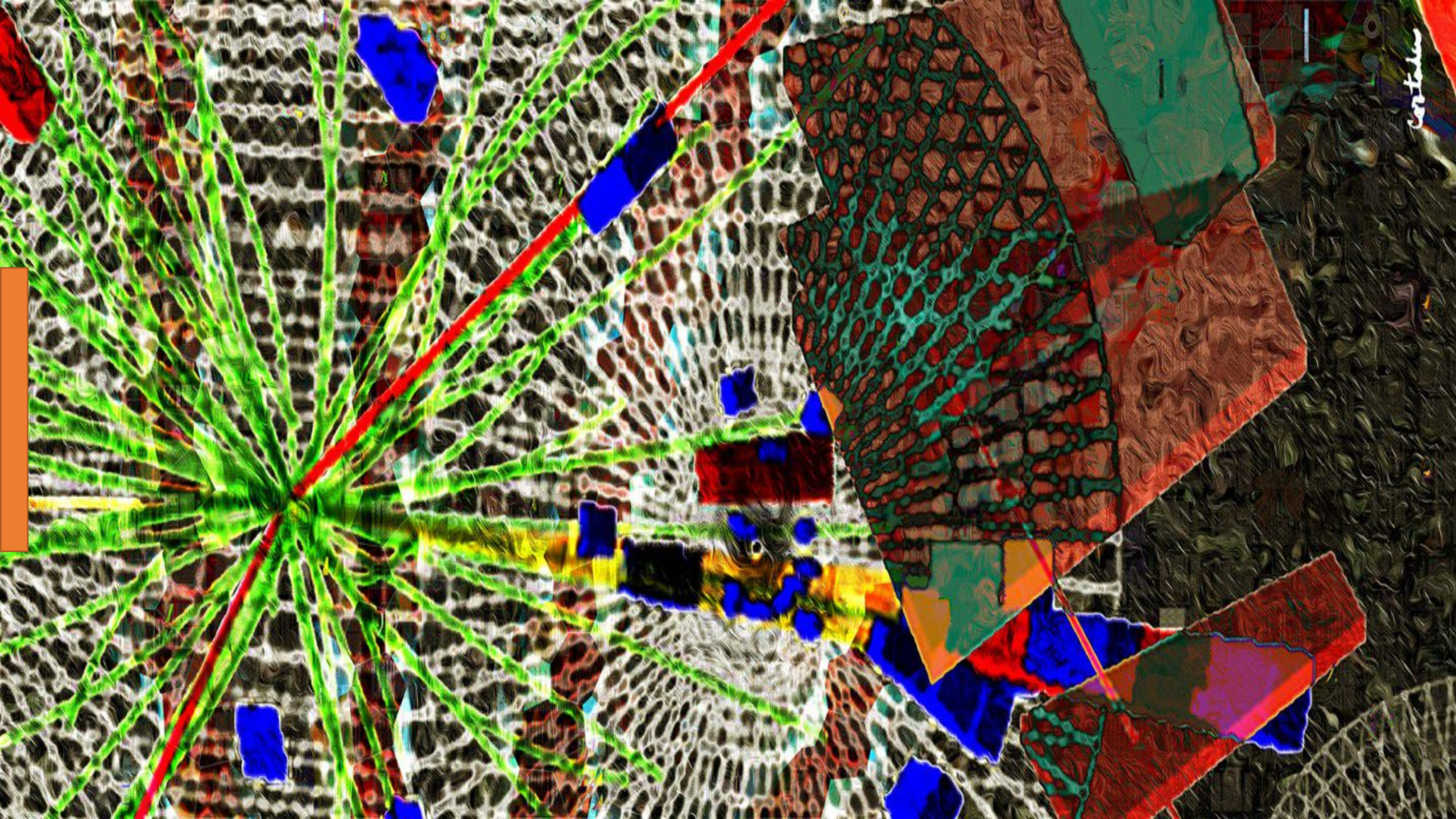


This clearly also impacts the scaling with the number of files, with a “less steep” increase of the execution time for cases in which we have more than one partition

Thanks Peter for the feedback!

Conclusions

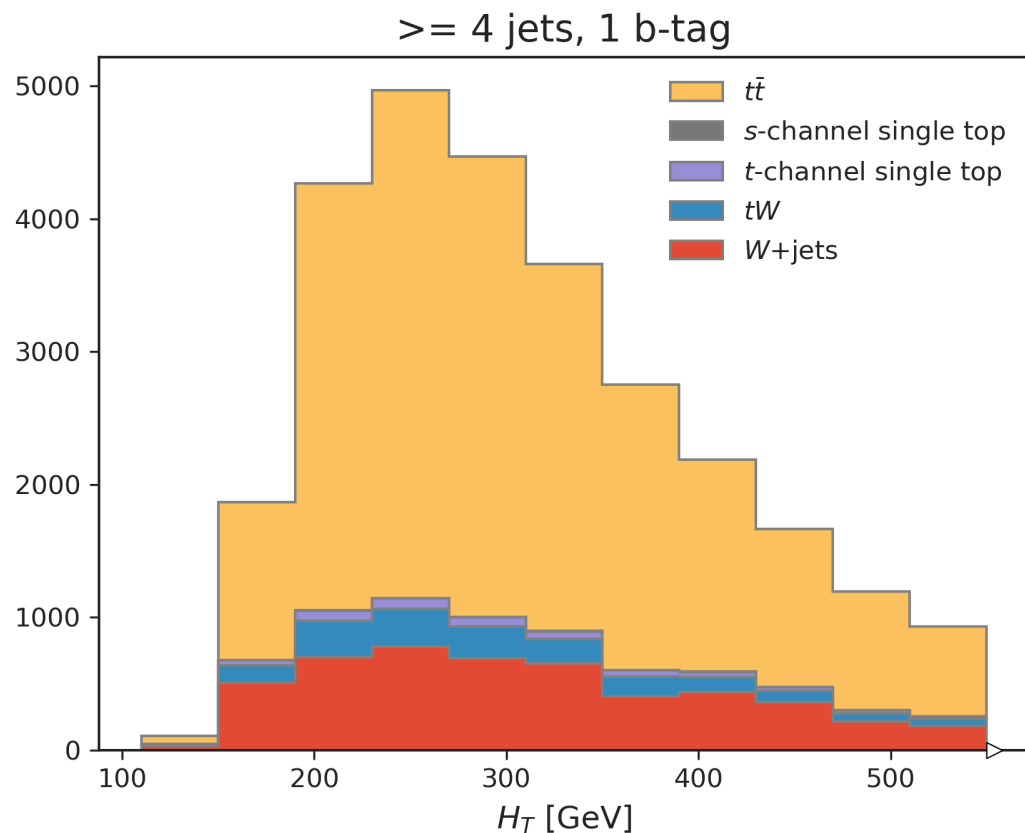
- User experience:
 - **Good**: for most of the systematics, almost no changes needed when migrating to coffea 2025
 - **Bad**: workarounds needed when numpy functions have to be used
- Performance:
 - The tests show how performance increased, however...
 - This is not yet representative of a full analysis (few systematics, did not run at full scale, etc.)!
 - Did not yet hit the case in which the scheduler chokes when building the task graph



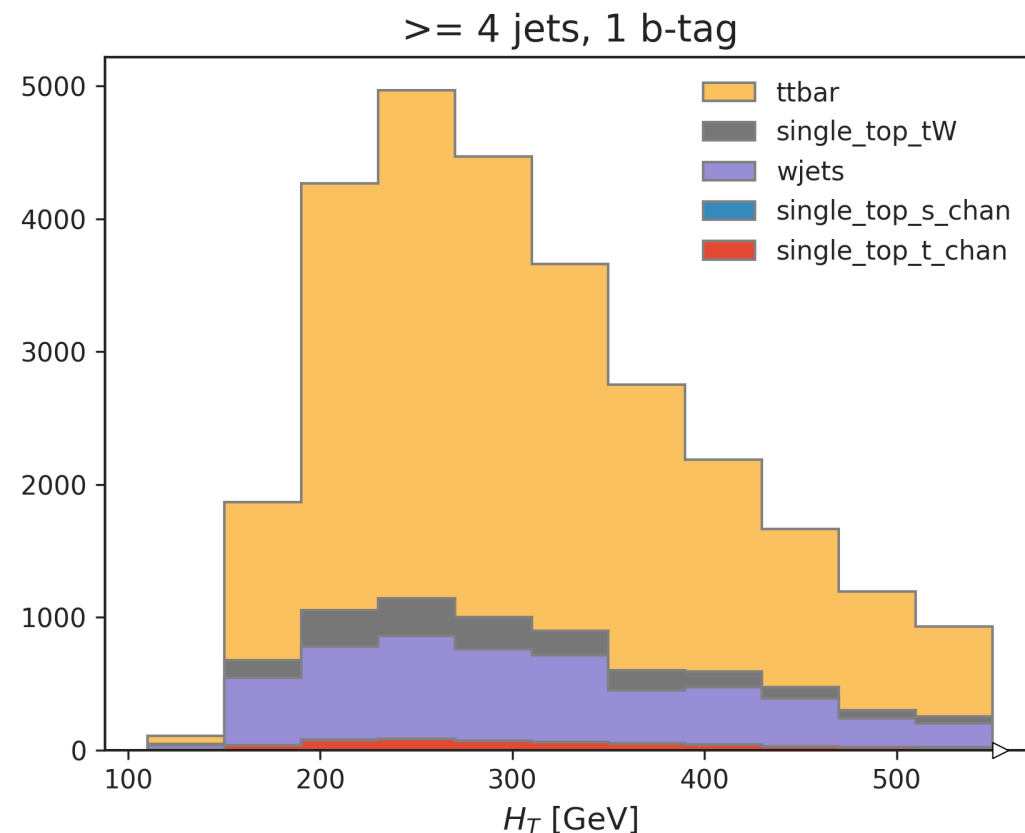
ARTIST

Histogram Comparison

New

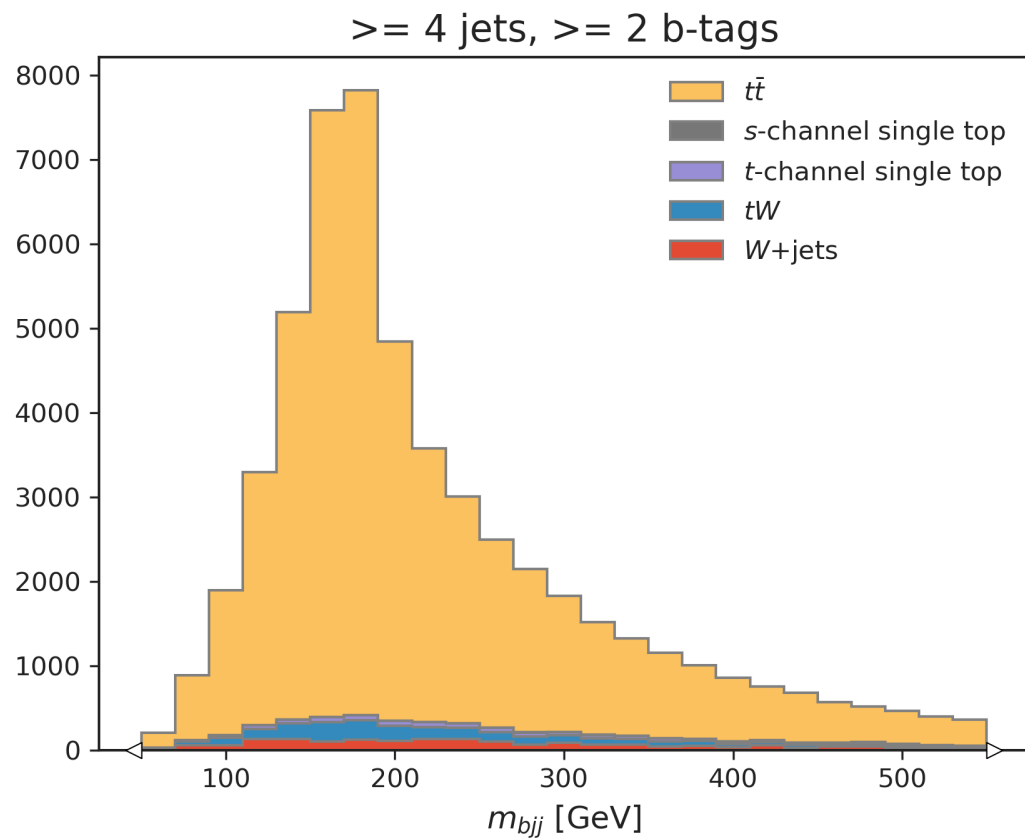


Old

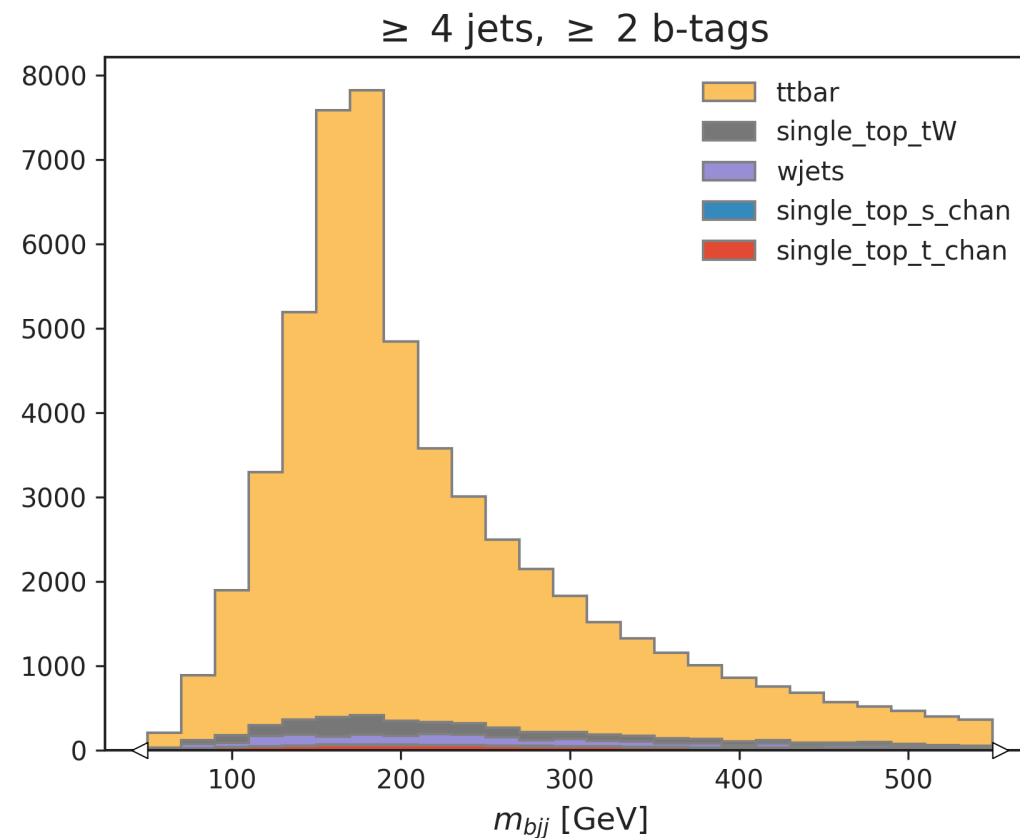


Histogram Comparison

New



Old

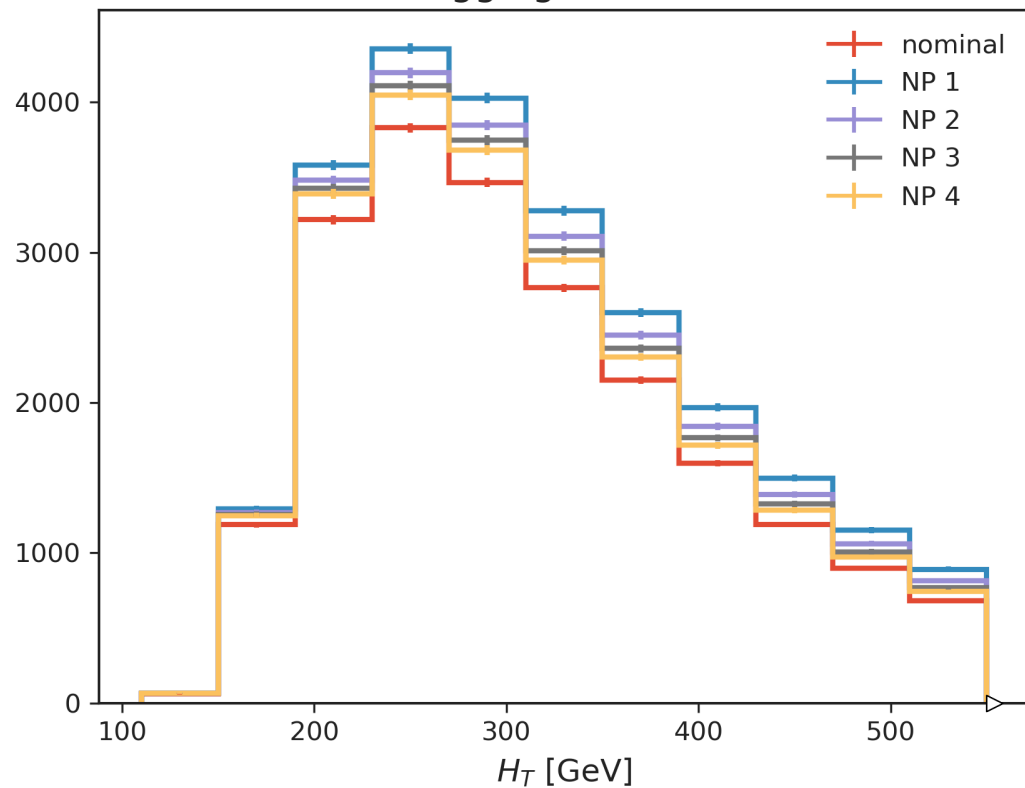


Histogram Comparison

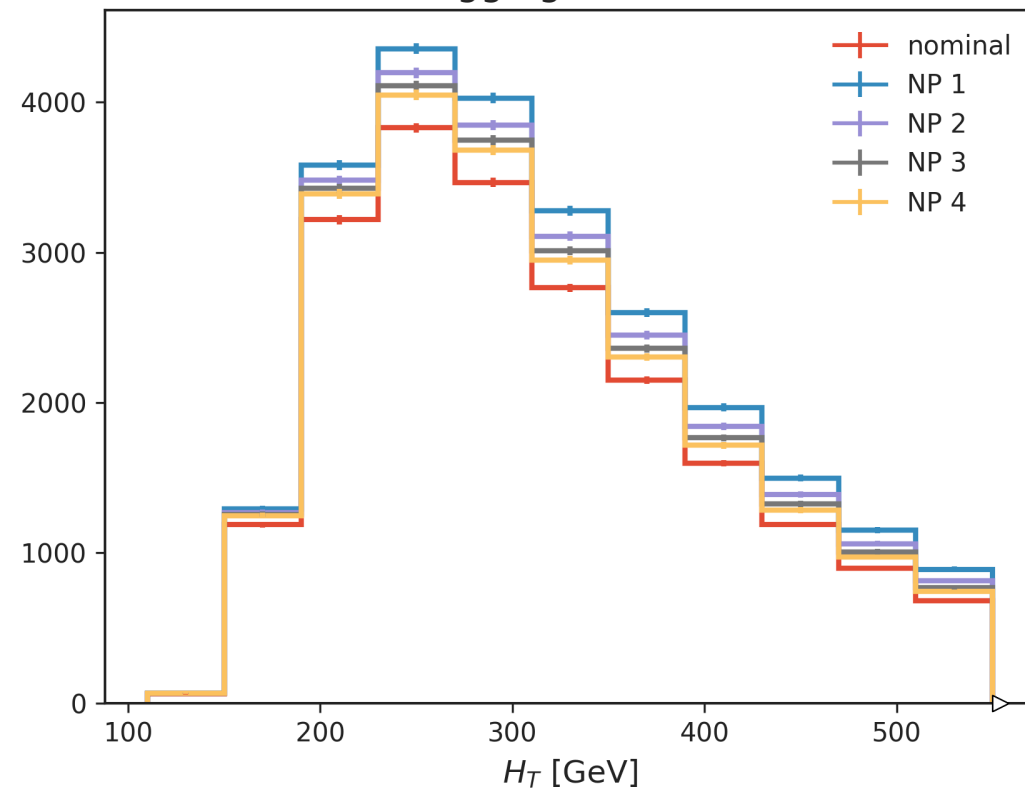
New

Old

b-tagging variations



b-tagging variations



Task Graphs

