ePIC SciGlass calorimeter benchmarks

Dmitry Kalinkin

April 20, 2023

Contents

1 License

2 Acknowledgements

We would thank the University of Kentucky Center for Computational Sciences and Information Technology Services Research Computing for their support and use of the Lipscomb Compute Cluster and associated research computing resources.

3 Computing setup

Record the software versions for posterity:

```
import os
print("\n".join(os.environ.get("buildInputs", "").split()))
```

```
/nix/store/pf4d1igmrypa9bzj3896jkk58zrpgnax-DD4hep-01-25-01
/nix/store/2bp5jzgzrj4di34vfavy0aw0nbws0ffr-Gaudi-36r5
/nix/store/2cscas1zr03177ns7k9i0i4mbbisj64n-podio-00-16-02
/nix/store/hs199hk2scmnm8rj6mmnlmwakhw3c3g6-EDM4hep-00-06
/nix/store/r2pzmsir1pvh4pyxk18qdvgis29bzkvc-EDM4eic-1.2.2
/nix/store/0qb782pjj9ga6cywy8smnlj6y418cily-epic-23.03.0
/nix/store/lmy3yaqbf4kb7wqr9c271daaxc9gc1g8-G4EMLOW-8.0
/nix/store/d2i8b0hjm642hggmh30fg35sbnyp3czc-G4ENSDFSTATE-2.3
/nix/store/d2i8b0hjm642hggmh30fg35sbnyp3czc-G4ENSDFSTATE-2.3
/nix/store/7a6zbab69xaf9sfdppppspk4kv8c6qh4-G4PARTICLEXS-4.0
/nix/store/rdg3q7y38dni0a8swzyvn5vjdn55lrgk-G4PhotonEvaporation-5.7
/nix/store/fc6r1qn0bvlbk703zafsgk4rrv6958hw-python3.10-awkward-1.9.0
/nix/store/8ja137vki0fkmsz6fc6r573y5p544lw1-python3.10-bokeh-2.4.3
/nix/store/nasgainr5as7qsshjfmr89ff2nfnl177-python3.10-boost-histogram-1.3.1
/nix/store/hq5h026bz1l2irvnl167a6zhpw5ikn2q-python3.10-dask-2022.7.0
/nix/store/8112a3mybp7d3vd5i0dbnf1lr3gg01z4-python3.10-dask-jobqueue-0.7.4
/nix/store/m3bz51dixv1kis08r3xwvczn19hf8vkx-python3.10-distributed-2022.7.0
/nix/store/w110cqs0alm1xxmcd8z5z7k5wygv1pph-python3.10-uproot-4.3.5
/nix/store/wqn525v598pjifnrqf6ic5ffk6xi0fwq-python3.10-setuptools-63.2.0
/nix/store/gbhqyy6kh56nfp7wcp04xwa9baarp0v5-python3.10-numpy-1.23.1
/nix/store/f429fw846j5ialc6fx34gak675alf8cp-python3.10-matplotlib-3.5.2
/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1
/nix/store/ipkvv26i5svl4rpqbprff1g54xpsjp5a-python3.10-scikit-learn-1.1.1
/nix/store/mpqnni2syc8hxgl3f0p228gagm0ky7c3-python3.10-catboost-1.1.1
/nix/store/gk08l2arg4wjjsf289sbw8nsp4xxgva5-python3.10-xgboost-1.5.2
/nix/store/2kh7vnwslrrnsv13z6pfj6lpdg1zirzi-python3.10-notebook-6.4.12
/nix/store/cx51iq79qjd18sz1fs9aw3cx7vnj1lfa-python3.10-joblib-1.1.0
/nix/store/10q1dmk53gqrqjxw9rcf8145ykr54nf0-python3.10-vector-0.11.0
```

We will be using Dask to run tasks on top of batch jobs:

```
import awkward as ak
import numpy as np
import vector
vector.register_awkward()

GeV = 1e0
MeV = 1e-3

NUM_ECALBARREL_SECTORS = 24
NUM_ECALBARREL_ROWS = 5

DELTA_PHI_PER_ROW = 2 * np.pi / NUM_ECALBARREL_SECTORS / NUM_ECALBARREL_ROWS

def eta2theta(eta):
    return 2 * np.arctan(np.exp(-eta))

def theta2eta(theta):
    return -np.log(np.tan(theta / 2))

import dask
from dask.distributed import Client, worker_client
import subprocess

try:
    slurm_version = subprocess.check_output(["sbatch", "--version"])
except:
    slurm_version = None

if slurm_version is not None:
    from dask_jobqueue import SLURMCluster
    if "cluster" not in globals():
        print("Cluster already initialized. Skipping initialization...")
        cluster = SLURMCluster()

        client = Client(cluster)

    NUM_EVENTS_DEFAULT = 10000
else:
    client = Client(processes=False)

    NUM_EVENTS_DEFAULT = 100
```

We will be using the `DETECTOR_` environment variables to locate the compact files:

```
from pathlib import Path
import os
DEFAULT_DETECTOR_PATH = Path(os.environ["DETECTOR_PATH"])
DEFAULT_DETECTOR_CONFIG = os.environ["DETECTOR_CONFIG"]
RESOURCE_DIR = Path(os.getcwd())
```

The `joblib.Memory` will be used to cache some of our computation. Note that the cache is read and written

on the side of the task, hence the path needs to be accessible over the network.

```
import joblib
memory = joblib.Memory(
  location=RESOURCE_DIR / "cache",
  verbose=0,
)

# This step is needed to enable ak.Array unpickling for joblib.Memory
from dask.distributed import WorkerPlugin
class AwkwardImporter(WorkerPlugin):
    def setup(self, worker):
        import awkward

client.register_worker_plugin(AwkwardImporter())
```

4 Detector simulation setup

```
@memory.cache
def prepare_calibrations_and_field_maps():
    import subprocess
    print(subprocess.check_output(
        ["ddsim",
         "--runType", "batch",
         "--numberOfEvents", "1",
         "--compactFile", DEFAULT_DETECTOR_PATH / (DEFAULT_DETECTOR_CONFIG + ".xml"),
         "--outputFile", "/tmp/nothing.edm4hep.root",
         "--enableGun",
        ],
        cwd=RESOURCE_DIR,
    ))

prepare_calibrations_and_field_maps()
```

```
from math import pi

@dask.delayed
@memory.cache
def run_ddsim_single(seed=1, particle="e-", p_min=2., p_max=2., theta_min=0.,
↪   theta_max=pi, phi_min=-pi, phi_max=pi, distribution="uniform",
↪   physics_list="FTFP_BERT", num_events=1000, branches=None,
↪   detector_path=DEFAULT_DETECTOR_PATH, detector_config=DEFAULT_DETECTOR_CONFIG,
↪   environ=None):
    assert seed != 0 # ddsim doesn't like a zero
    import shutil
    import tempfile
    import subprocess
    with tempfile.TemporaryDirectory() as workdir:
        workdir = Path(workdir)
```

```
        output_path = workdir / "sim.edm4hep.root"
        shutil.copytree(RESOURCE_DIR / "calibrations", workdir / "calibrations",
        ↪  symlinks=True)
        shutil.copytree(RESOURCE_DIR / "fieldmaps", workdir / "fieldmaps",
        ↪  symlinks=True)
        shutil.copytree(detector_path, workdir / "detector", symlinks=True)
        subprocess.check_output(
            ["ddsim",
             "--runType", "batch",
             "--filter.tracker", "edep0",
             "-v", "WARNING",
             "--numberOfEvents", f"{num_events}",
             "--compactFile",  workdir / "detector" / (detector_config + ".xml"),
             "--outputFile", output_path,
             "--physicsList", physics_list,
             "--enableGun",
             "--gun.momentumMin", f"{p_min}*GeV",
             "--gun.momentumMax", f"{p_max}*GeV",
             "--gun.thetaMin", f"{theta_min}",
             "--gun.thetaMax", f"{theta_max}",
             "--gun.phiMin", f"{phi_min}",
             "--gun.phiMax", f"{phi_max}",
             "--gun.particle", particle,
             "--gun.distribution", distribution,
             "--random.seed", str(seed),
            ],
            cwd=workdir,
            env=dict(environ if environ is not None else os.environ,
            ↪  DETECTOR_PATH=workdir / "detector")
            )
        if branches is None:
            branches = [
                "EcalBarrelSciGlassHits.cellID",
                "EcalBarrelSciGlassHits.energy",
                "EcalBarrelSciGlassHits.position.x",
                "EcalBarrelSciGlassHits.position.y",
                "EcalBarrelSciGlassHits.position.z",
                "MCParticles.PDG",
                "MCParticles.generatorStatus",
                "MCParticles.momentum.x",
                "MCParticles.momentum.y",
                "MCParticles.momentum.z",
                "MCParticles.simulatorStatus",
                "MCParticles.vertex.x",
                "MCParticles.vertex.y",
                "MCParticles.vertex.z",
            ]
        import uproot
        events = uproot.open(output_path)["events"].arrays(branches)
        return events


@dask.delayed
```

```
def run_ddsim_single_abseta(num_events=None, abseta_min=None, abseta_max=None,
↪  **kwargs):
    if (abseta_min is not None) or (abseta_max is not None):
        if (kwargs.get("theta_min", None) is not None) or (kwargs.get("theta_max",
        ↪  None) is not None):
            raise ValueError("Specifying \"abseta\" and \"theta\" cuts at the same time
            ↪  is not supported")
        if abseta_max is None:
            abseta_max = np.inf
        if (abseta_min is None) or (abseta_min == 0.):
            results = [
                run_ddsim_single(**dict(kwargs,
                    num_events=num_events,
                    theta_min=eta2theta(-abseta_max),
                    theta_max=eta2theta(abseta_max),
                )),
            ]
        else:
            results = [
                run_ddsim_single(**dict(kwargs,
                    num_events=num_events // 2,
                    theta_min=eta2theta(-abseta_max),
                    theta_max=eta2theta(-abseta_min),
                )),
                run_ddsim_single(**dict(kwargs,
                    num_events=num_events // 2,
                    theta_min=eta2theta(abseta_min),
                    theta_max=eta2theta(abseta_max),
                )),
            ]
    else:
        results = [
            run_ddsim_single(**dict(kwargs,
                num_events=num_events,
            )),
        ]
    with worker_client() as client:
        results = client.gather(client.compute(results))
    return ak.concatenate(results)


@dask.delayed
def run_ddsim(seed=1, num_events=1000, num_events_single=1000, **kwargs):
    num_batches = int(np.ceil(num_events / float(num_events_single)))
    nums_events = [num_events_single] * (num_batches - 1) + [num_events -
    ↪  num_events_single * (num_batches - 1)]
    seeds = range(seed, seed + num_batches)
    batches = [
        run_ddsim_single_abseta(seed=seed, num_events=num_events, **kwargs)
        for num_events, seed in zip(nums_events, seeds)
    ]
    with worker_client() as client:
        batches = client.gather(client.compute(batches))
```

```
    return ak.concatenate(batches)
```

4.1 Alternative detector configurations

The alternative detector configurations are enabled by creating creating a modified copy of the "compact" directory. The resulting directory is cached in order to avoid unnecessary re-runs of the simulation by using its cache.

```
def hashdir(path):
    import hashlib
    digest = hashlib.md5()

    def recurse(cur_path, digest=None):
        paths = cur_path.iterdir()
        for file_ in sorted([p for p in paths if p.is_file()]):
            digest.update(str(file_.relative_to(path)).encode())
            with open(file_, "rb") as fp:
                digest.update(fp.read())

        paths = cur_path.iterdir()
        for dir_ in sorted([p for p in paths if p.is_dir()]):
            recurse(dir_, digest)

    recurse(path, digest)

    return digest.hexdigest()


def with_detector_path(ops):
    """
    Parameters
    ----------
    ops : Path -> none
    Function to be executed on the copied path
    """
    def func(base_detector_path=None, *args, **kwargs):
        assert base_detector_path is not None

        import shutil
        import tempfile

        conf_dir = RESOURCE_DIR / "configurations"
        conf_dir.mkdir(exist_ok=True)
        detector_path = Path(tempfile.mkdtemp(dir=conf_dir))
        shutil.copytree(base_detector_path, detector_path, symlinks=True,
        ↪  dirs_exist_ok=True)

        ops(detector_path, *args, **kwargs)

        hash_ = hashdir(detector_path)
        output_dir = conf_dir / hash_
        print(f"Output directory: {output_dir}")
```

```
        if output_dir.exists():
            print(f"Replacing existing path!")
            def chmod_add_write(func, path, _):
                import stat
                Path(path).chmod(stat.S_IRUSR | stat.S_IWUSR)
                Path(path).parent.chmod(stat.S_IRWXU)
                func(path)
            shutil.rmtree(output_dir, onerror=chmod_add_write)
        detector_path.rename(output_dir)

        return output_dir

    return func


def modify_xml(xml_path, ops_nodes):
    """
    Parameters
    ----------
    ops_nodes : xml.etree.ElementTree.ElementTree -> xml.etree.ElementTree.ElementTree
    Function to be executed on the copied path
    """
    from lxml import etree

    with open(xml_path, "r") as fp:
        root = etree.parse(fp)

    root = ops_nodes(root)

    import os
    import stat
    os.chmod(xml_path, stat.S_IRUSR | stat.S_IWUSR)
    root.write(xml_path)
```

```
@with_detector_path
def remove_carbon_fiber(detector_path):
    def ops_nodes(root):
        node_root = root.getroot()

        node_carbon_fiber_support, =
        ↪  node_root.xpath("detectors/detector/sectors/carbon_fiber_support")
        node_carbon_fiber_support.getparent().remove(node_carbon_fiber_support)

        return root

    modify_xml(detector_path / "compact" / "ecal" / "barrel_sciglass.xml", ops_nodes)


without_carbon_fiber = remove_carbon_fiber(DEFAULT_DETECTOR_PATH)
```

```
Output directory: /home/dka268/epic-ecal-sciglass/configurations/424754e1e3d4a200b7c84552915c5d1d
```

```
Replacing existing path!
```

```python
@with_detector_path
def remove_wedge_box(detector_path):
    def ops_nodes(root):
        node_root = root.getroot()

        node_wedge_box, = node_root.xpath("detectors/detector/wedge_box")
        node_wedge_box.getparent().remove(node_wedge_box)

        return root

    modify_xml(detector_path / "compact" / "ecal" / "barrel_sciglass.xml", ops_nodes)


without_wedge_box = remove_wedge_box(DEFAULT_DETECTOR_PATH)
without_supports = remove_wedge_box(without_carbon_fiber)
```

```
Output directory: /home/dka268/epic-ecal-sciglass/configurations/4332ef41f9f2e1cf438e78fcb9720f54
Replacing existing path!
Output directory: /home/dka268/epic-ecal-sciglass/configurations/b92a53b9a66017808dc20b3932300ebe
Replacing existing path!
```

```python
@with_detector_path
def remove_gaps_longitudinal(detector_path):
    def ops_nodes(root):
        node_root = root.getroot()

        node_rows, = node_root.xpath("detectors/detector/sectors/rows")
        node_dimensions, = list(node_rows.iter("dimensions"))
        node_dimensions.attrib["gap"] = "0 * mm"

        nodes_family = list(node_rows.iter("family"))
        for node in nodes_family:
            node.attrib["y1"] = "2.05 * cm"

        return root

    modify_xml(detector_path / "compact" / "ecal" / "barrel_sciglass.xml", ops_nodes)


without_gaps_longitudinal = remove_gaps_longitudinal(without_supports)
```

```
Output directory: /home/dka268/epic-ecal-sciglass/configurations/b93ef6cc8301e472df3be8c7be853f88
Replacing existing path!
```

```python
@with_detector_path
def remove_gaps_transverse_tower(detector_path):
    def ops_nodes(root):
```

```
        node_root = root.getroot()

        node_rows, = node_root.xpath("detectors/detector/sectors/rows")
        nodes_family = list(node_rows.iter("family"))
        for node in nodes_family:
            node.attrib["x1"] = "2.1 * cm"

        return root

    modify_xml(detector_path / "compact" / "ecal" / "barrel_sciglass.xml", ops_nodes)


without_gaps_longitudinal_and_transverse_tower =
↪  remove_gaps_transverse_tower(without_gaps_longitudinal)
```

```
Output directory: /home/dka268/epic-ecal-sciglass/configurations/a958ff96c9127ad9183f5649a22e6a72
Replacing existing path!
```

```
@with_detector_path
def remove_gaps_transverse_sector(detector_path):
    def ops_nodes(root):
        node_root = root.getroot()

        node_rows, = node_root.xpath("detectors/detector/sectors/rows")
        node_rows.attrib["deltaphi"] = "pi / 12 / 5"

        nodes_family = list(node_rows.iter("family"))
        for node in nodes_family:
            node.attrib["x1"] = "2.2 * cm"

        return root

    modify_xml(detector_path / "compact" / "ecal" / "barrel_sciglass.xml", ops_nodes)


without_gaps =
↪  remove_gaps_transverse_sector(without_gaps_longitudinal_and_transverse_tower)
```

```
Output directory: /home/dka268/epic-ecal-sciglass/configurations/57067bc4a51d08c273dcb6aa9a7a8367
Replacing existing path!
```

The following implements a facility needed to adjust the Birks' constant

```
@with_detector_path
def set_sciglass_birks_constant(detector_path, value="0.0333*mm/MeV"):
    def ops_nodes(root):
        node_root = root.getroot()

        node_sciglass_birks_constant, =
        ↪  node_root.xpath("material[@name='SciGlass']/constant[@name='BirksConstant']")
        node_sciglass_birks_constant.attrib["value"] = value
```

```
        return root

    modify_xml(detector_path / "compact" / "materials.xml", ops_nodes)


without_sciglass_birks_constant = set_sciglass_birks_constant(DEFAULT_DETECTOR_PATH,
↪  "0*mm/MeV")
```

```
Output directory: /home/dka268/epic-ecal-sciglass/configurations/03fca6148bdf8e5190e337ad5e695635
Replacing existing path!
```

```
@with_detector_path
def endcapn_switch_to_sciglass(detector_path):
    def ops_nodes(root):
        node_root = root.getroot()

        node_crystal_length, =
        ↪  node_root.xpath("define/constant[@name='EcalEndcapN_CrystalModule_length']")
        node_crystal_length.attrib["value"] = "450.00*mm"

        node_crystal, =
        ↪  node_root.xpath("detectors/detector/placements/disk_12surface/crystal")
        node_crystal.attrib["material"] = "SciGlass"

        return root

    modify_xml(detector_path / "compact" / "ecal" / "backward_PbWO4.xml", ops_nodes)


endcapn_sciglass = endcapn_switch_to_sciglass(DEFAULT_DETECTOR_PATH)
```

```
Output directory: /home/dka268/epic-ecal-sciglass/configurations/4eb5af222b3fda89bf983f93d7373b5f
Replacing existing path!
```

5 Plotting setup

```
import matplotlib as mpl
import matplotlib.pyplot as plt

def setup_presentation_style():
    mpl.rcParams.update(mpl.rcParamsDefault)
    plt.style.use('ggplot')
    plt.rcParams.update({
        'axes.labelsize': 8,
        'axes.titlesize': 9,
        'figure.titlesize': 9,
        'figure.figsize': (4, 3),
```

```
        'legend.fontsize': 7,
        'xtick.labelsize': 8,
        'ytick.labelsize': 8,
        'pgf.rcfonts': False,
    })

setup_presentation_style()
```

6 Data analysis

6.1 Pion rejection factor

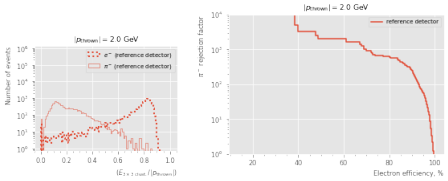
6.1.1 Analysis setup

sim_settings

```
common_sim_params = dict(
    num_events=NUM_EVENTS_DEFAULT,
    theta_min=0.5,
    theta_max=2.8,
)

def edep_sum_classifier(events):
    p_thrown = np.sqrt(
        events["MCParticles.momentum.x"][:,0] ** 2
        + events["MCParticles.momentum.y"][:,0] ** 2
        + events["MCParticles.momentum.z"][:,0] ** 2
    )
    return ak.sum(events["EcalBarrelSciGlassHits.energy"], axis=1) /  p_thrown

edep_sum_classifier.xlabel = r"$(E_{\mathrm{dep.}} / |p_{\mathrm{thrown}}|)$"


<<find_cluster_def>>


def edep_cluster_sum_classifier(size=3):
    def classifier(events):
        clusters = find_cluster(events, size=size)
        p_thrown = np.sqrt(
            events["MCParticles.momentum.x"][:,0] ** 2
            + events["MCParticles.momentum.y"][:,0] ** 2
            + events["MCParticles.momentum.z"][:,0] ** 2
        )
        return ak.sum(ak.flatten(clusters.egrid, axis=-1), axis=1) /  p_thrown

    classifier.xlabel = r"$(E_{" + str(size) + r"\times " + str(size) +
    ↪  r"\:\mathrm{clust.}} / |p_{\mathrm{thrown}}|)$"

    return classifier


def apply_energy_threshold(events, threshold):
```

```
    cond = events["EcalBarrelSciGlassHits.energy"] > (threshold / GeV)
    return ak.Array({
        field: (events[field][cond] if field.startswith("EcalBarrelSciGlassHits.") else
        ↪  events[field])
        for field in events.fields
    })


def enable_energy_threshold(classifier, threshold):
    def new_classifier(events):
        return classifier(apply_energy_threshold(events, threshold))
    if hasattr(classifier, "xlabel"):
        new_classifier.xlabel = classifier.xlabel
    return new_classifier

energies = [0.05, 0.1, 0.2, 0.5, 1., 2., 3., 5., 8., 10., 15., 18.]
```

```
p_thrown = 2. # GeV
particles = ["e-", "pi-"]

<<sim_settings>>

sim_settings = [
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
        ),
        label="reference detector",
        visual=dict(
            color="C0",
        ),
    ),
]

output_dir = Path("results/test")

sim = {}
<<book_sim>>
<<pion_rej>>
```



book_sim

```
sim.setdefault(p_thrown, {})
for ix, setting in enumerate(sim_settings):
```

```
    sim_params = setting["sim"]
    sim[p_thrown][ix] = {
        particle: client.compute(run_ddsim(particle=particle, p_min=p_thrown,
        ↪  p_max=p_thrown, **sim_params))
        for particle in particles
    }
```

pion_rej

```
import numpy as np

output_dir.mkdir(parents=True, exist_ok=True)

bins_e_over_p = np.linspace(0., 1.01, 102)

fig = plt.figure(figsize=np.array(mpl.rcParams["figure.figsize"]) * 0.8)

y_pred = {}
for ix, setting in enumerate(sim_settings):
    classifier = setting["classifier"]
    label = setting["label"]
    visual_params = setting["visual"]
    y_pred.setdefault(ix, {})
    y_pred[ix]["e-"] = classifier(sim[p_thrown][ix]["e-"].result())
    y_pred[ix]["pi-"] = classifier(sim[p_thrown][ix]["pi-"].result())
    _, _, _ = plt.hist(y_pred[ix]["e-"], bins=bins_e_over_p, label=f"$e^-$ ({label})",
    ↪  histtype="step", lw=2, ls=":", **visual_params)
    _, _, _ = plt.hist(y_pred[ix]["pi-"], bins=bins_e_over_p, label=f"$\pi^-$
    ↪  ({label})", histtype="step", **visual_params)

plt.title(rf"$|p_{{\mathrm{{thrown}}}}| = {p_thrown}$ GeV")
plt.xlabel(" or ".join(set([setting["classifier"].xlabel for setting in
↪  sim_settings])), loc="right")
plt.ylabel("Number of events", loc="top")
plt.minorticks_on()
plt.savefig(f"e_over_p_{p_thrown:.2f}.pdf")
plt.yscale("log")
plt.ylim(top=plt.ylim()[1] * 1000)
if len(sim_settings) > 5:
    plt.legend(bbox_to_anchor=(0, 1.05), loc="lower left")
else:
    plt.legend()
plt.savefig(output_dir / f"e_over_p_{p_thrown:.2f}_log.pdf", bbox_inches="tight")
plt.show()

fig = plt.figure(figsize=np.array(mpl.rcParams["figure.figsize"]) * 0.8)

fpr = {}
tpr = {}
for ix, setting in enumerate(sim_settings):
    label = setting["label"]
    visual_params = setting["visual"]

    from sklearn.metrics import roc_curve
```

```
    fpr[ix], tpr[ix], _ = roc_curve(
        np.concatenate([np.ones_like(y_pred[ix]["e-"]),
        ↪  np.zeros_like(y_pred[ix]["pi-"])]),
        np.concatenate([y_pred[ix]["e-"], y_pred[ix]["pi-"]]),
    )
    cond = fpr[ix] != 0 # avoid infinite rejection (region of large uncertainty)
    cond &= tpr[ix] != 1 # avoid linear interpolation (region of large uncertainty)
    plt.plot(tpr[ix][cond] * 100, 1 / fpr[ix][cond], label=label, **visual_params)

plt.title(rf"$|p_{{\mathrm{{thrown}}}}| = {p_thrown}$ GeV")
plt.xlabel("Electron efficiency, %", loc="right")
plt.ylabel("$\pi^-$ rejection factor", loc="top")
plt.yscale("log")
plt.ylim(1., max(settings["sim"].get("num_events", NUM_EVENTS_DEFAULT) for settings in
↪  sim_settings))
plt.minorticks_on()
if len(sim_settings) > 5:
    plt.legend(bbox_to_anchor=(0, 1.05), loc="lower left")
else:
    plt.legend()
plt.savefig(output_dir / f"pi-_rej_{p_thrown:.2f}.pdf", bbox_inches="tight")
plt.show()
```

pion_rej_vs_pt

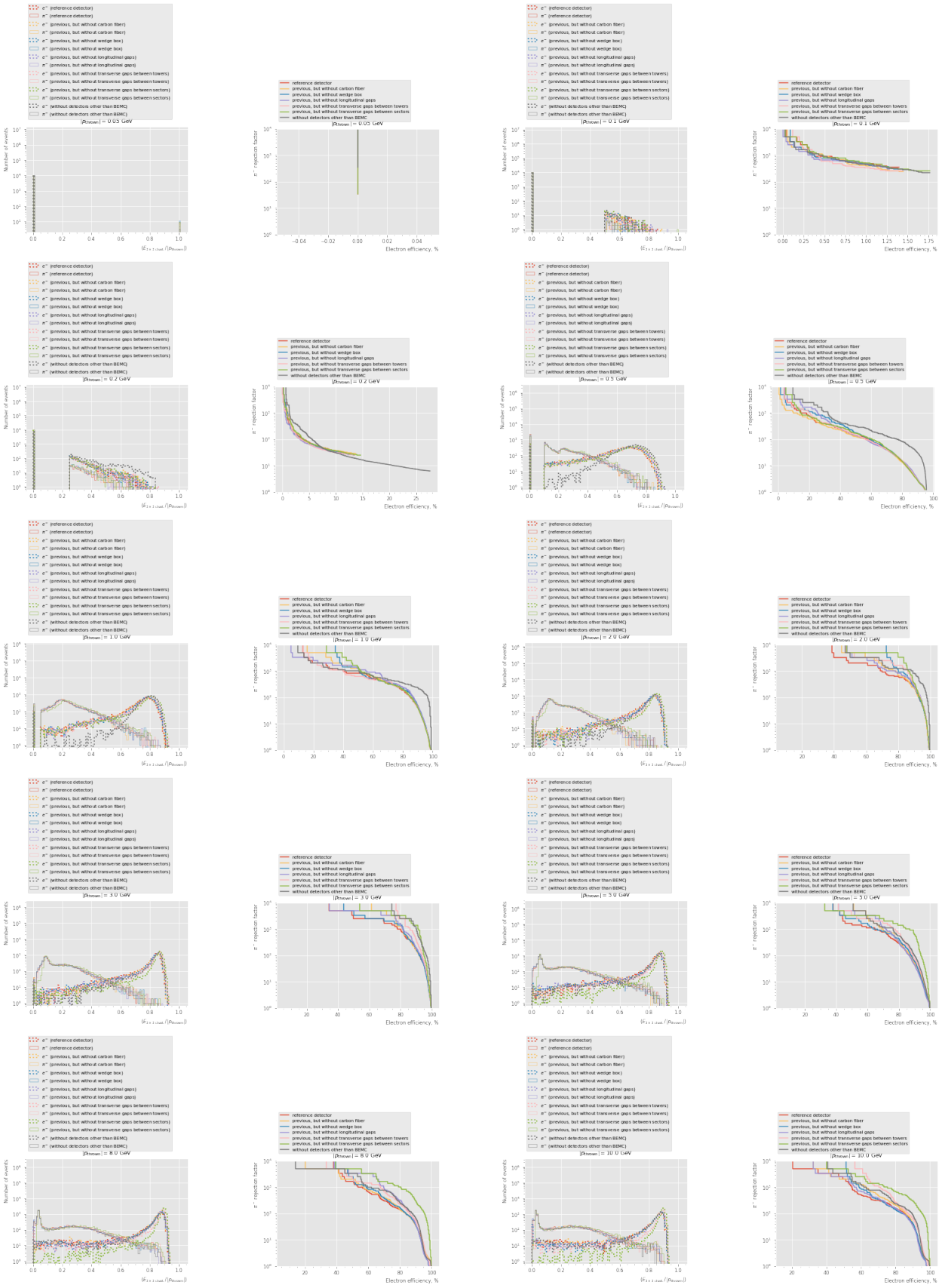
```
particles = ["e-", "pi-"]

sim = {}
for p_thrown in energies:
    <<book_sim>>

roc = {}
for p_thrown in energies:
    <<pion_rej>>

    for ix, setting in enumerate(sim_settings):
        def mk_interp(tpr, fpr):
            def interp(eff):
                return np.interp(eff, tpr, fpr)
            return interp
        roc.setdefault(ix, []).append(mk_interp(tpr[ix], fpr[ix]))

for ix, setting in enumerate(sim_settings):
    label = setting["label"]
    visual_params = setting["visual"]
    plt.plot(energies, 1 / np.array([fpr(0.95) for fpr in roc[ix]]), label=f"{label} at
    ↪  95% efficiency", ls="-", lw=2. - ix * 0.15, **visual_params)

plt.xlabel(r"$|p_{\mathrm{thrown}}|$, GeV", loc="right")
plt.ylabel(r"$\pi^-$ rejection factor", loc="top")
plt.yscale("log")
plt.ylim(1e0, 1e3)
if len(sim_settings) > 5:
    plt.legend(bbox_to_anchor=(0, 1.05), loc="lower left")
```

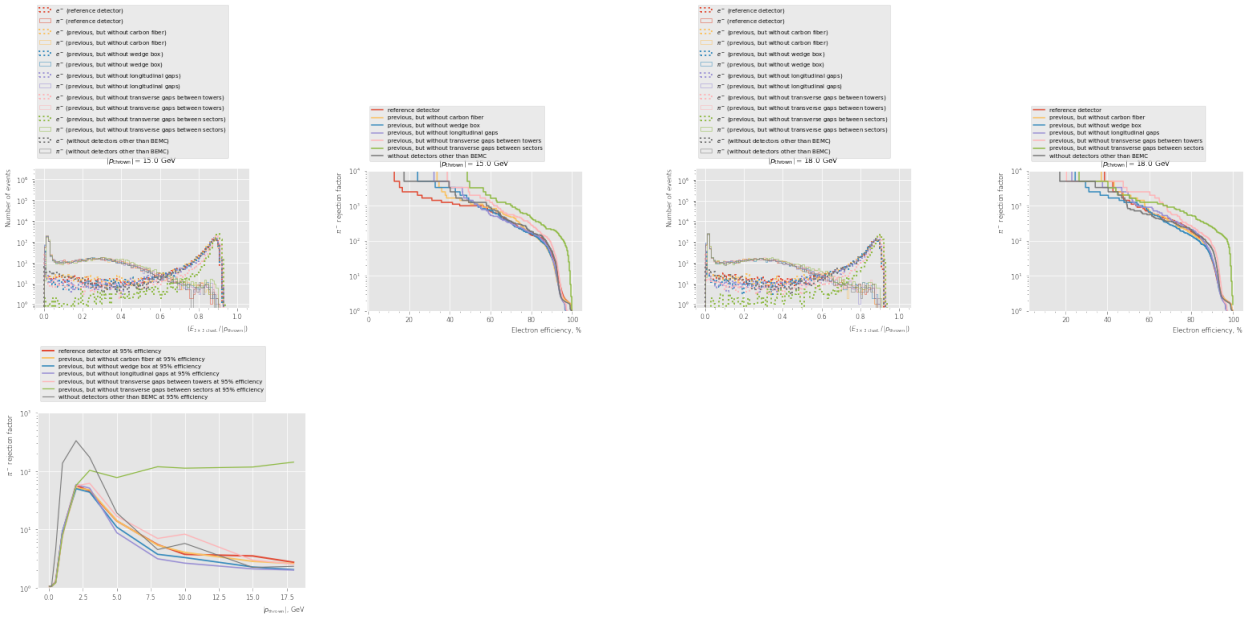
```
else:
    plt.legend()
plt.savefig(output_dir / "pi-_rej.pdf", bbox_inches="tight")
plt.show()
```

6.1.2 Detector variations

```
<<sim_settings>>

sim_settings = [
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
        ),
        label="reference detector",
        visual=dict(
            color="C0",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
            detector_path=without_carbon_fiber,
        ),
        label="previous, but without carbon fiber",
        visual=dict(
            color="C4",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
            detector_path=without_supports,
        ),
        label="previous, but without wedge box",
        visual=dict(
            color="C1",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
            detector_path=without_gaps_longitudinal,
```

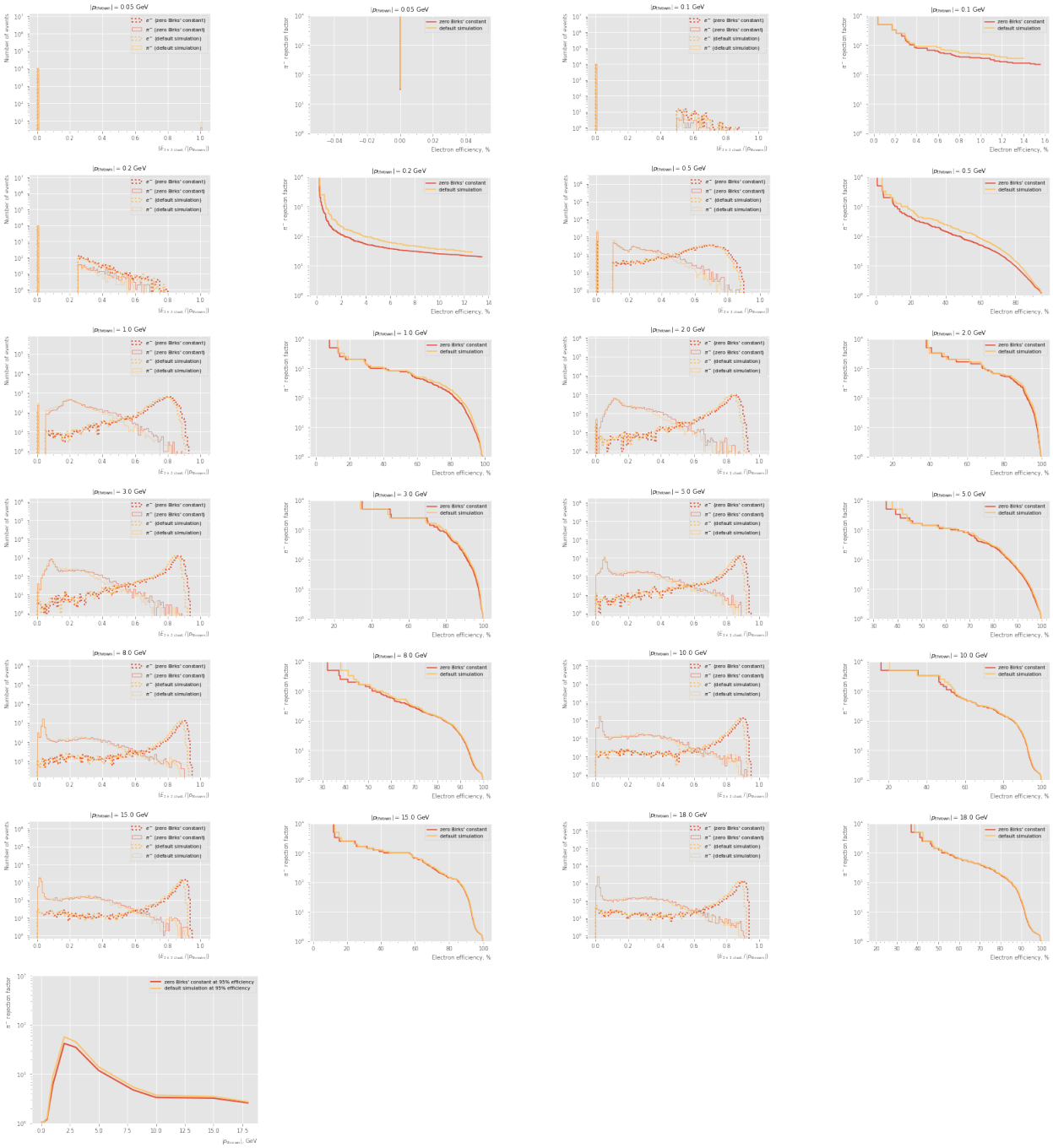
```
        ),
        label="previous, but without longitudinal gaps",
        visual=dict(
            color="C2",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
            detector_path=without_gaps_longitudinal_and_transverse_tower,
        ),
        label="previous, but without transverse gaps between towers",
        visual=dict(
            color="C6",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
            detector_path=without_gaps,
        ),
        label="previous, but without transverse gaps between sectors",
        visual=dict(
            color="C5",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
            detector_config="epic_sciglass_only",
        ),
        label="without detectors other than BEMC",
        visual=dict(
            color="C3",
        ),
    ),
]

output_dir = Path("results/detector_variants")

<<pion_rej_vs_pt>>
```





6.1.3 Effect of Birks constant

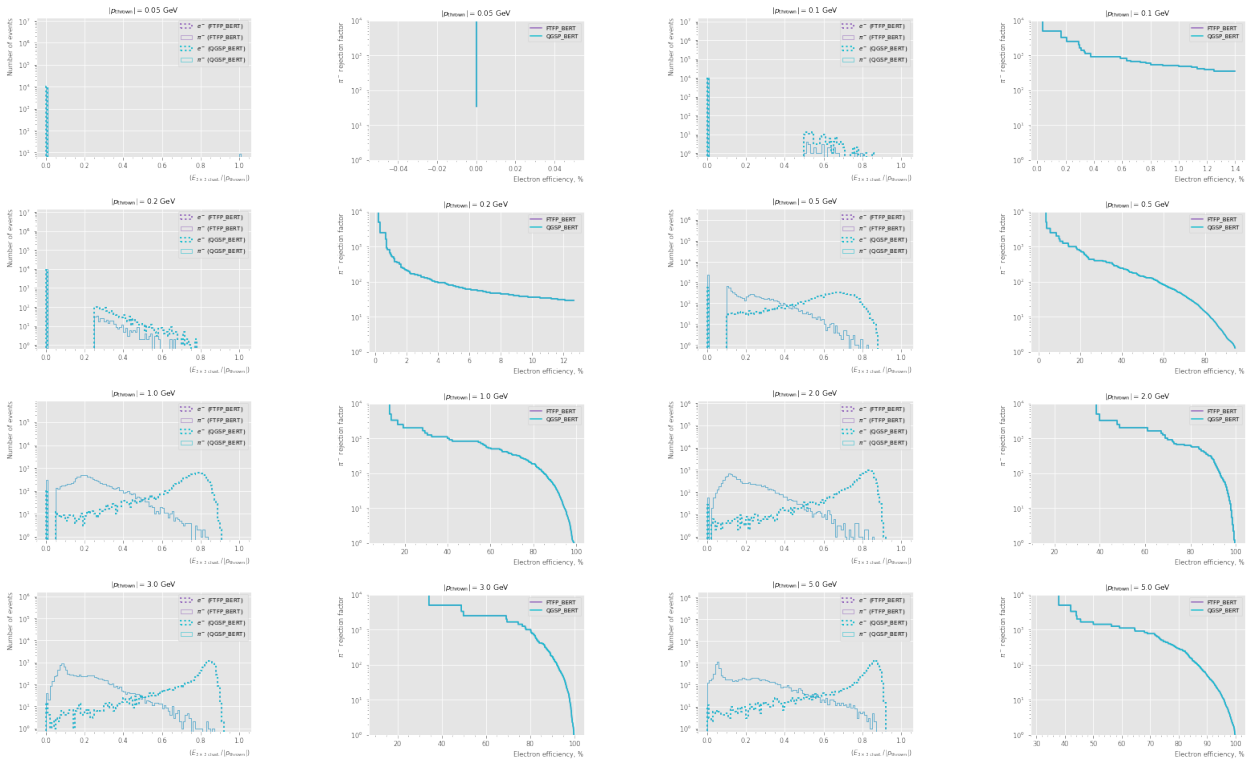
```
<<sim_settings>>
sim_settings = [
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
            detector_path=without_sciglass_birks_constant,
        ),
        label="zero Birks' constant",
        visual=dict(
            color="C0",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
        ),
        label="default simulation",
        visual=dict(
            color="C4",
        ),
    ),
]

output_dir = Path("results/birks_constant_variations")

<<pion_rej_vs_pt>>
```

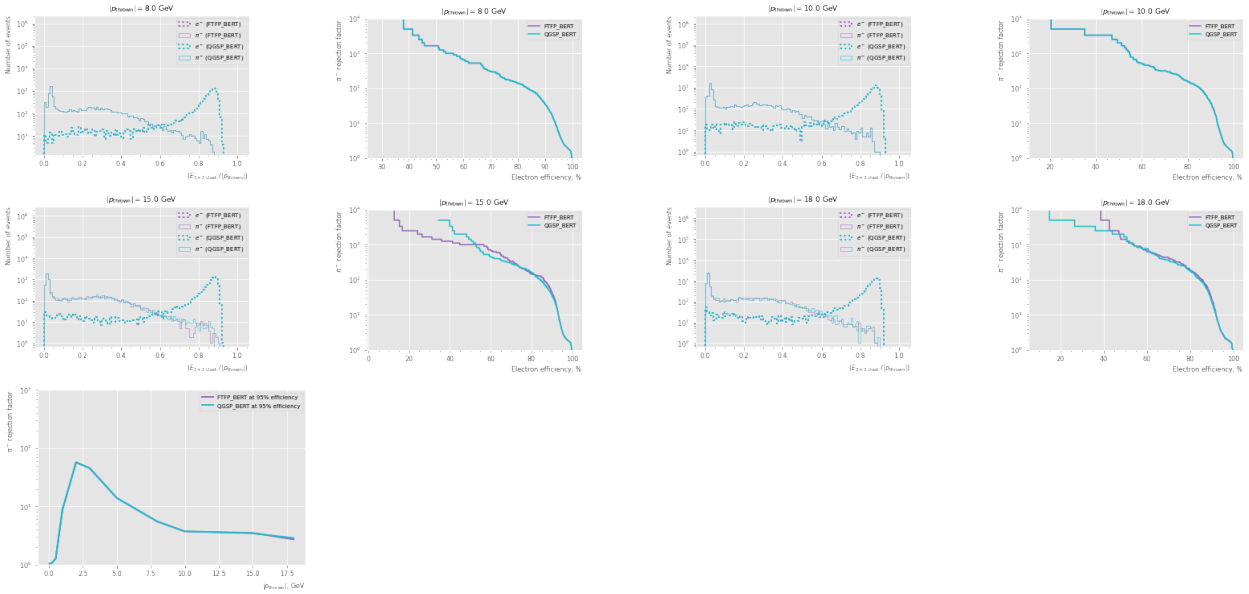


6.1.4 Systematic uncertainty

```
<<sim_settings>>

sim_settings = [
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
```

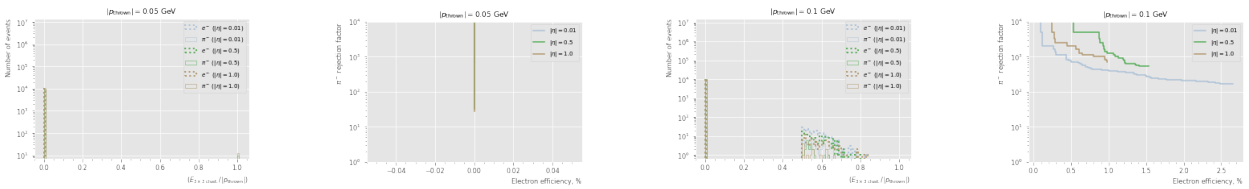
```
            common_sim_params,
            physics_list="FTFP_BERT"
        ),
        label=r"FTFP_BERT",
        visual=dict(
            color="tab:purple",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
            physics_list="QGSP_BERT"
        ),
        label=r"QGSP_BERT",
        visual=dict(
            color="tab:cyan",
        ),
    ),
]

output_dir = Path("results/physics_list")

<<pion_rej_vs_pt>>
```

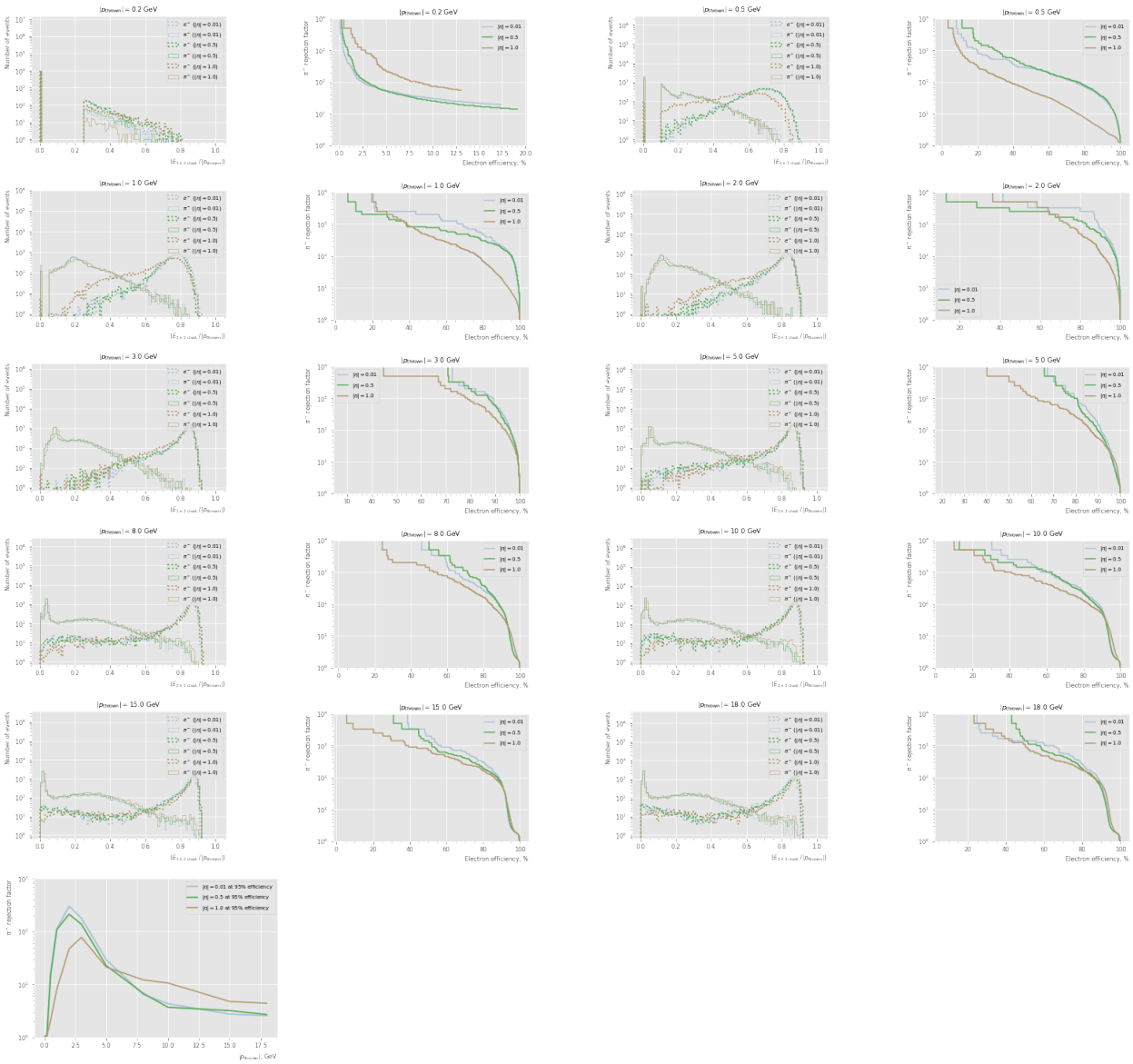




6.1.5 Rapidity bins

```
<<sim_settings>>

sim_settings = [
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
            abseta_min=eta,
            abseta_max=eta,
            theta_min=None,
            theta_max=None,
        ),
        label=rf"$|\eta| = {eta}$",
        visual=dict(
            color=color,
        ),
    ) for eta, color in zip([0.01, 0.5, 1.], list(mpl.colors.XKCD_COLORS))
]

output_dir = Path("results/charge_eta_bins")

<<pion_rej_vs_pt>>
```





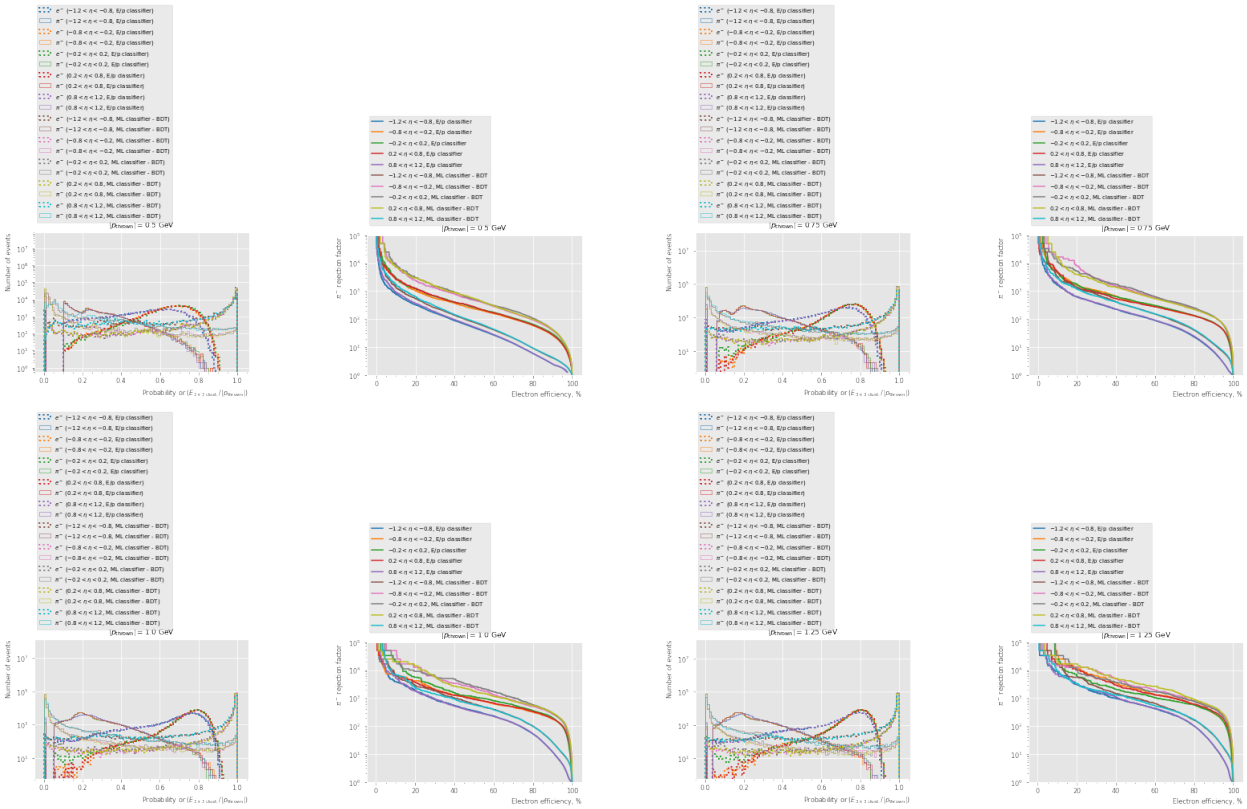
6.1.6 For table

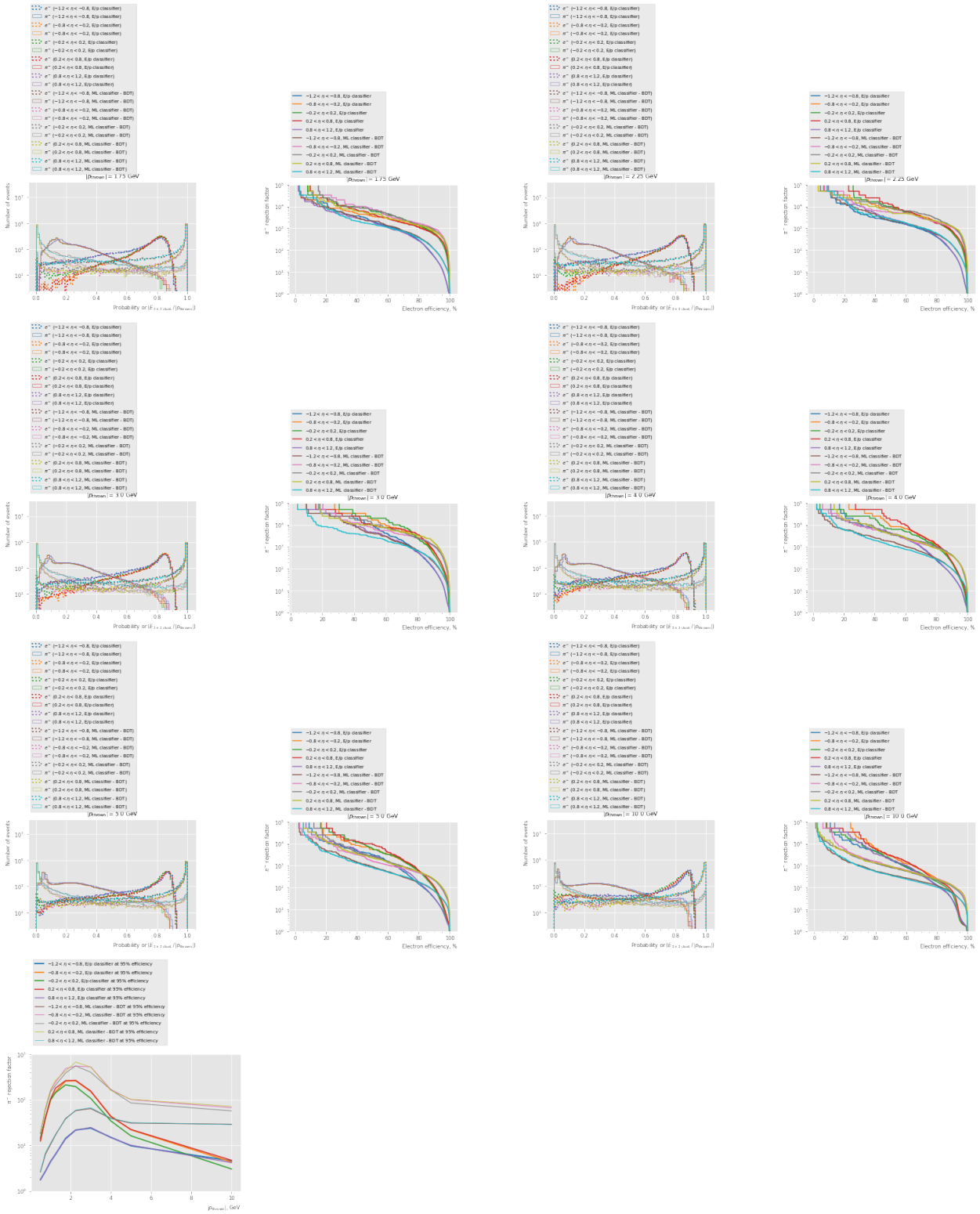
```
<<ml_sim_settings>>

particles = ["e-", "pi-"]

#energies = [0.26, 0.31, 0.39, 0.48, 0.58, 0.72, 0.89, 1.09, 1.34, 1.65, 2.03, 2.49,
↪  3.07, 3.77, 4.64, 5.71, 7.02, 8.64]
#eta_bins = [-1., 0., 1.]

#energies = [0.50, 0.75, 1.00, 2.25, 5.00, 10.00]
#eta_bins = [-1.2, -0.8, -0.2, 0.2, 0.8, 1.2]

energies = [0.50, 0.75, 1.00, 1.25, 1.75, 2.25, 3.00, 4.00, 5.00, 10.00]
eta_bins = [-1.2, -0.8, -0.2, 0.2, 0.8, 1.2]
```

```
sim_settings[0]["classifier"] = edep_sum_classifier # disable shower profiling

sim = {}
for p_thrown in energies:
    <<book_sim>>

sim_train = sim
sim = {}

from xgboost import XGBClassifier

<<train_def>>
<<to_x_with_3momentum_def>>

xgboost_with_p_vec_classifier = train(
    [sim_train[p_thrown][0]["e-"] for p_thrown in energies],
    [sim_train[p_thrown][0]["pi-"] for p_thrown in energies],
    enable_energy_threshold(to_x_with_3momentum, 50 * MeV),
    XGBClassifier(),
)

sim_settings = [
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
            num_events=NUM_EVENTS_DEFAULT * 10,
            theta_min=eta2theta(eta_low),
            theta_max=eta2theta(eta_high),
        ),
        label=rf"${eta_low:.1f} < \eta < {eta_high:.1f}$, E/p classifier",
        visual=dict(
            color=color,
        ),
    ) for eta_low, eta_high, color in zip(eta_bins, eta_bins[1:],
    ↪  list(mpl.colors.TABLEAU_COLORS))
] + [
    dict(
        classifier=enable_energy_threshold(xgboost_with_p_vec_classifier, 50 * MeV),
        sim=dict(
            common_sim_params,
            num_events=NUM_EVENTS_DEFAULT * 10,
            theta_min=eta2theta(eta_low),
            theta_max=eta2theta(eta_high),
        ),
        label=rf"${eta_low:.1f} < \eta < {eta_high:.1f}$, ML classifier - BDT",
        visual=dict(
            color=color,
        ),
    ) for eta_low, eta_high, color in zip(eta_bins, eta_bins[1:],
    ↪  list(mpl.colors.TABLEAU_COLORS)[len(eta_bins) - 1:])
```

```
]

output_dir = Path("results/for_Barak")

<<pion_rej_vs_pt>>

for row in np.array(list(zip(*[1 / np.array([fpr(0.95) for fpr in roc[ix]]) for ix,
↪  setting in enumerate(sim_settings)]))):
    print("\t".join(map(str, row)))
```

```
/project/rhfate2_uksr/nix/store/gk0812arg4wjjsf289sbw8nsp4xxgva5-python3.10-xgboost-1.5.2/lib/python3.1
  from pandas import MultiIndex, Int64Index
/project/rhfate2_uksr/nix/store/m3bz51dixv1kis08r3xwvczn19hf8vkx-python3.10-distributed-2022.7.0/lib/py
  (XGBClassifier(base_score=None, booster=None, cols ... , 1., 1., 0.]))
Consider scattering large objects ahead of time
with client.scatter to reduce scheduler burden and
keep data on workers

    future = client.submit(func, big_data)    # bad

    big_future = client.scatter(big_data)     # good
    future = client.submit(func, big_future)  # good
  warnings.warn(
```





1.7753160062491122 12.531328320802006 14.39055979277594 12.655024044545684
1.7506696311339087 2.6292264815691224 18.331805682859763 18.32508704416346
18.86080724254998 2.6252748334591307
2.7190907360578622 39.95205753096285 40.74979625101874 40.0 2.705408110813516
6.439979392065945 64.89292667099286 61.576354679802954 64.30868167202573

```
7.004763239002521
4.401602183194683 97.75171065493646 99.50248756218906 103.84215991692628
4.476075377109351 10.491528091066465 161.29032258064515 149.92503748125938
163.39869281045753 11.090163025396473
6.375518010838381 150.37593984962407 143.67816091954023 181.1594202898551
6.4645419872002075 16.998130205677377 242.71844660194174 209.20502092050208
267.379679144385 17.639795378373613
14.285714285714285 258.39793281653743 213.67521367521366 264.55026455026456
13.787398317937406 38.94080996884735 485.43689320388347 389.1050583657588
432.9004329004329 37.67897513187641
21.791239921551536 262.4671916010499 196.46365422396858 269.54177897574124
21.50537634408602 58.20721769499418 552.4861878453039 558.659217877095 675.6756756756757
59.41770647653
24.1196333815726 153.60983102918587 108.34236186348862 156.98587127158555
24.900398406374503 64.51612903225806 531.9148936170213 404.8582995951417
526.3157894736842 67.98096532970767
15.126304643775525 44.24778761061947 34.638032559750606 43.27131112072696
14.865467518953471 39.41663381947182 169.49152542372883 166.9449081803005
162.07455429497568 39.714058776806986
9.855129594954173 22.084805653710248 16.270745200130165 22.49212775528565
10.141987829614605 31.01736972704715 101.31712259371834 85.6898029134533
103.09278350515464 31.786395422759057
4.610632117663331 4.345181194055792 3.049803287687944 4.738887309259786 4.202563563773902
29.2654375182909 67.43088334457181 57.43825387708214 72.09805335255948 28.785261945883708
```

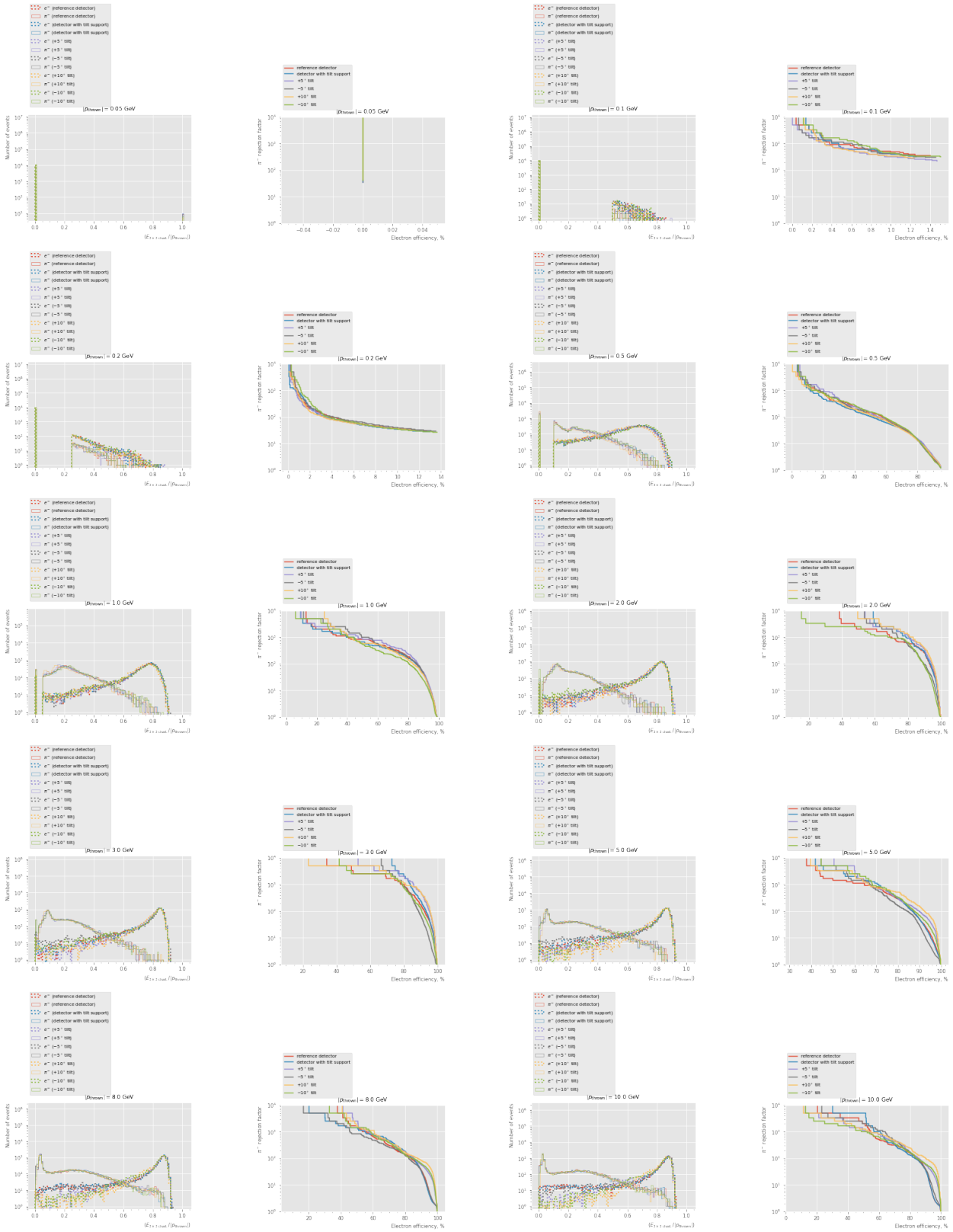
6.1.7 Study of effect of the phi tilt

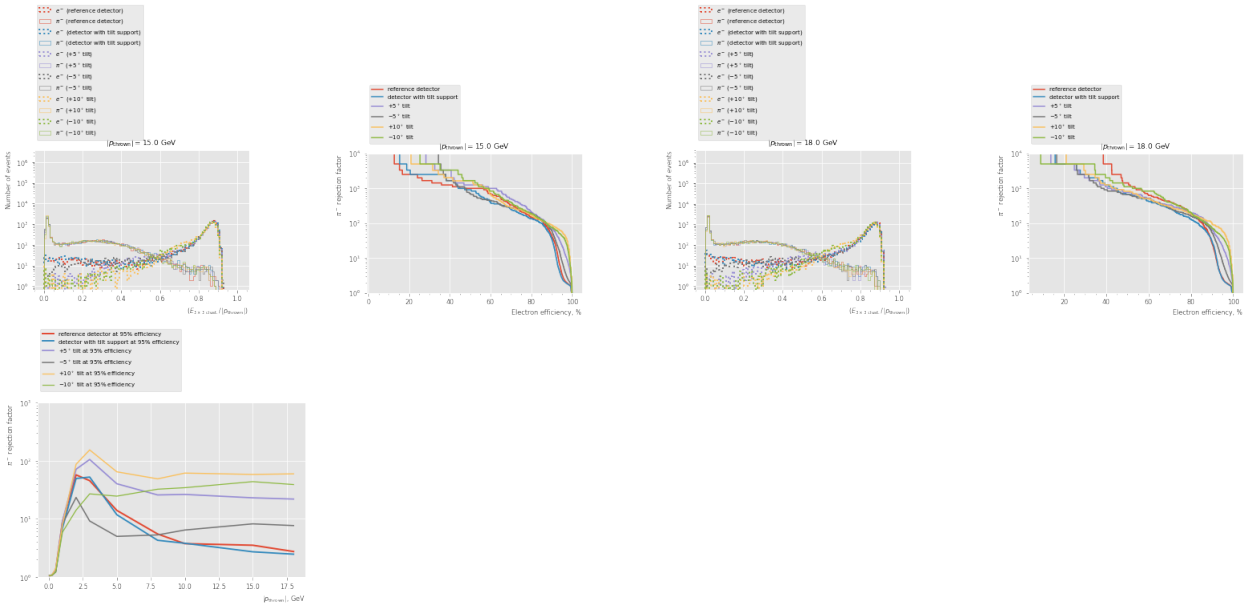
```
<<sim_settings>>


import json

@memory.cache
def custom_environ(epic_git=None):
    return json.loads(subprocess.check_output([
        "nix", "develop", ".#epic",
        "--override-input", "epic-git", f"github:eic/epic/{epic_git}",
        "-c", "python3", "-c", "import os, json; print(json.dumps(dict(os.environ)))",
    ]))


environ_with_tilt = custom_environ(epic_git="pr/phi_tilt")


@with_detector_path
def set_tilt(detector_path, tilt):
    def ops_nodes(root):
        node_root = root.getroot()

        node_sectors, = node_root.xpath("detectors/detector/sectors")
        node_sectors.attrib["phi_tilt"] = f"{tilt} * degree"

        return root
```

```
    modify_xml(detector_path / "compact" / "ecal" / "barrel_sciglass.xml", ops_nodes)


sim_settings = [
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
        ),
        label="reference detector",
        visual=dict(
            color="C0",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
            detector_path=environ_with_tilt["DETECTOR_PATH"],
            environ=environ_with_tilt,
        ),
        label="detector with tilt support",
        visual=dict(
            color="C1",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
            detector_path=set_tilt(environ_with_tilt["DETECTOR_PATH"], 5),
            environ=environ_with_tilt,
        ),
        label="$+5^\circ$ tilt",
        visual=dict(
            color="C2",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
            detector_path=set_tilt(environ_with_tilt["DETECTOR_PATH"], -5),
            environ=environ_with_tilt,
        ),
        label="$-5^\circ$ tilt",
        visual=dict(
            color="C3",
```

```
        ),
    ),
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
            detector_path=set_tilt(environ_with_tilt["DETECTOR_PATH"], 10),
            environ=environ_with_tilt,
        ),
        label="$+10^\circ$ tilt",
        visual=dict(
            color="C4",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
            detector_path=set_tilt(environ_with_tilt["DETECTOR_PATH"], -10),
            environ=environ_with_tilt,
        ),
        label="$-10^\circ$ tilt",
        visual=dict(
            color="C5",
        ),
    ),
]

output_dir = Path("results/phi_tilt")

<<pion_rej_vs_pt>>
```

```
warning: Git tree '/home/dka268/epic-ecal-sciglass' is dirty
warning: not writing modified lock file of flake 'git+file:///home/dka268/epic-ecal-sciglass':
 Updated input 'epic-git':
    'github:eic/epic/5c5d4ace5198276106bf294643b2ab2ba3f6eb74' (2023-03-18)
   'github:eic/epic/864fb6904be948a5ffe89a9493c216c525c176bf' (2023-03-30)
trace: qqq
Output directory: /home/dka268/epic-ecal-sciglass/configurations/85da4603364a664c6b4bf8a3375e5896
Replacing existing path!
Output directory: /home/dka268/epic-ecal-sciglass/configurations/2174f7b867e834998e2363b1c962705c
Replacing existing path!
Output directory: /home/dka268/epic-ecal-sciglass/configurations/54d07c26f7cf2a94aaa3060ca4db88d2
Replacing existing path!
Output directory: /home/dka268/epic-ecal-sciglass/configurations/6efc7120f2773fdf2d922a63f5f679c5
Replacing existing path!
```


e⁻ (reference detector)
π⁻ (reference detector)
e⁻ (detector with tilt support)
π⁻ (detector with tilt support)
e⁻ (+5° tilt)
π⁻ (+5° tilt)
e⁻ (−5° tilt)
π⁻ (−5° tilt)
e⁻ (+10° tilt)
π⁻ (+10° tilt)
e⁻ (−10° tilt)
π⁻ (−10° tilt)
reference detector
detector with tilt support
+5° tilt
−5° tilt
+10° tilt
−10° tilt
|p_thrown| = 0.05 GeV
|p_thrown| = 0.1 GeV
|p_thrown| = 0.2 GeV
|p_thrown| = 0.5 GeV
|p_thrown| = 1.0 GeV
|p_thrown| = 2.0 GeV
|p_thrown| = 3.0 GeV
|p_thrown| = 5.0 GeV
|p_thrown| = 8.0 GeV
|p_thrown| = 10.0 GeV
Number of events
π⁻ rejection factor
Electron efficiency, %

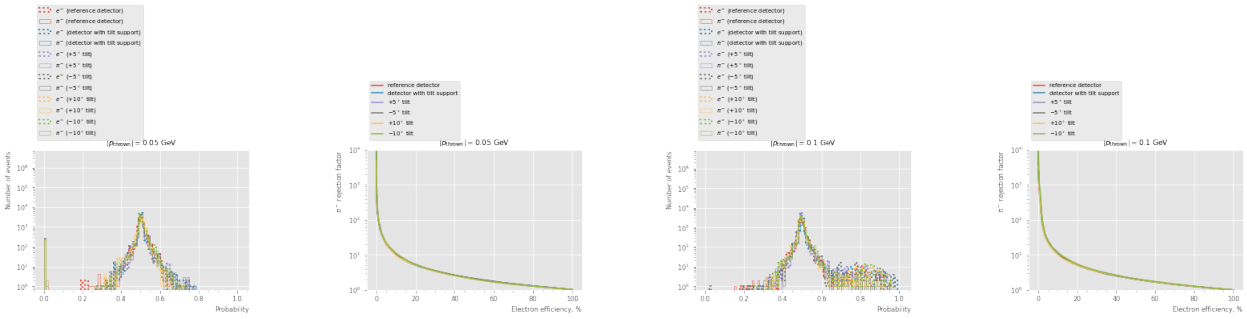


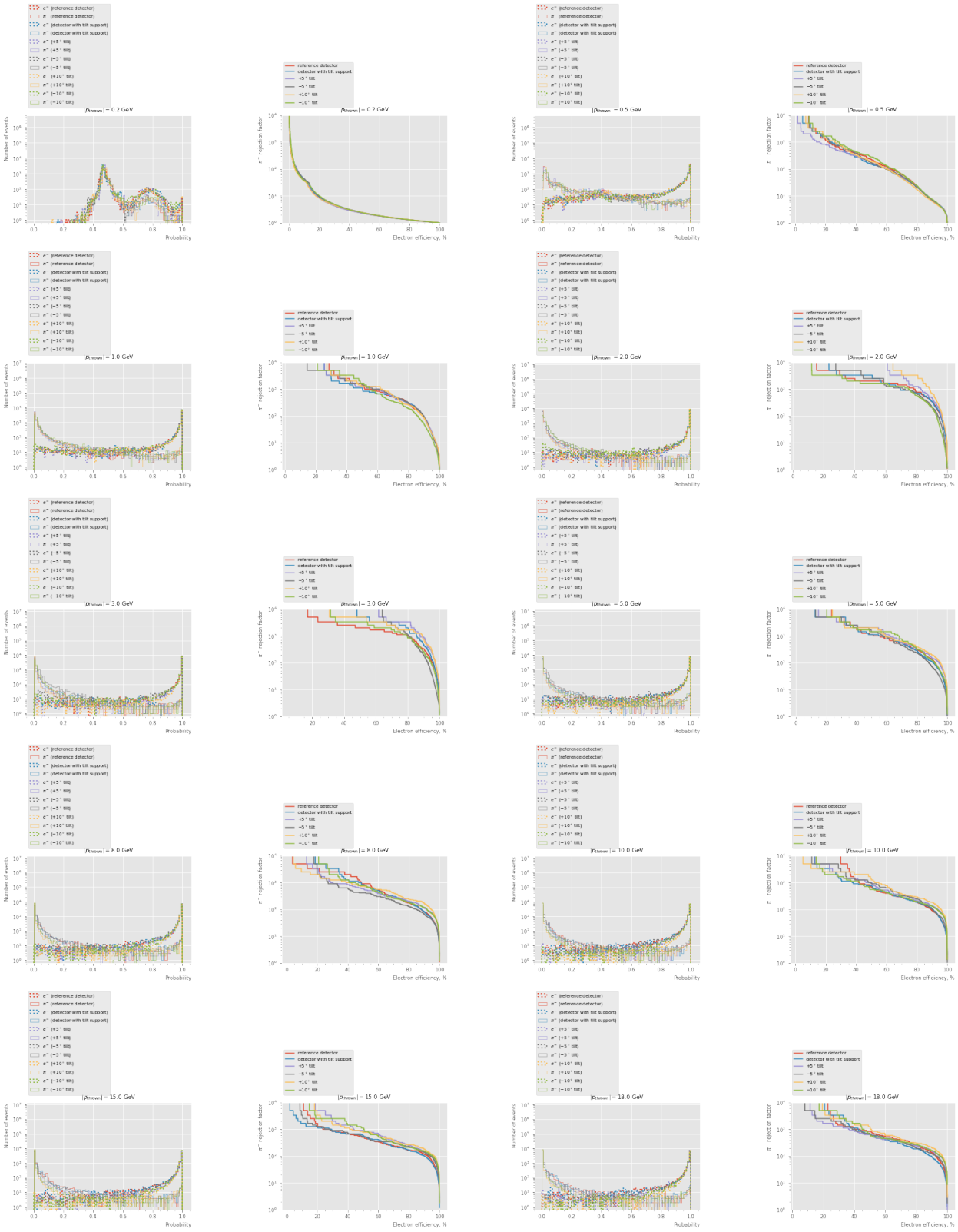
```
<<ml_sim_settings>>

particles = ["e-", "pi-"]

sim_settings = [
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
        ),
        label="reference detector",
        visual=dict(
            color="C0",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
            detector_path=environ_with_tilt["DETECTOR_PATH"],
            environ=environ_with_tilt,
        ),
        label="detector with tilt support",
        visual=dict(
            color="C1",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
```

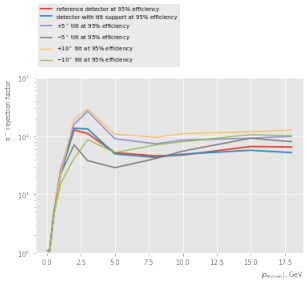
```
            common_sim_params,
            detector_path=set_tilt(environ_with_tilt["DETECTOR_PATH"], 5),
            environ=environ_with_tilt,
        ),
        label="$+5^\circ$ tilt",
        visual=dict(
            color="C2",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
            detector_path=set_tilt(environ_with_tilt["DETECTOR_PATH"], -5),
            environ=environ_with_tilt,
        ),
        label="$-5^\circ$ tilt",
        visual=dict(
            color="C3",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
            detector_path=set_tilt(environ_with_tilt["DETECTOR_PATH"], 10),
            environ=environ_with_tilt,
        ),
        label="$+10^\circ$ tilt",
        visual=dict(
            color="C4",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
            detector_path=set_tilt(environ_with_tilt["DETECTOR_PATH"], -10),
            environ=environ_with_tilt,
        ),
        label="$-10^\circ$ tilt",
        visual=dict(
            color="C5",
        ),
    ),
]

for setting in sim_settings:
    setting["sim"]["seed"] = 31337 # training seed
```

```
sim = {}
for p_thrown in energies:
    <<book_sim>>

sim_train = sim
sim = {}

from xgboost import XGBClassifier

<<train_def>>
<<to_x_with_3momentum_def>>

xgboost_with_p_vec_classifier_tilts = [
    train(
        [sim_train[p_thrown][ix]["e-"] for p_thrown in energies],
        [sim_train[p_thrown][ix]["pi-"] for p_thrown in energies],
        enable_energy_threshold(to_x_with_3momentum, 50 * MeV),
        XGBClassifier(),
    ) for ix, setting in enumerate(sim_settings)
]

for setting, classifier in zip(sim_settings, xgboost_with_p_vec_classifier_tilts):
    setting["classifier"] = classifier
    setting["sim"]["seed"] = 1 # evaluation seed

output_dir = Path("results/phi_tilt_ml")

<<pion_rej_vs_pt>>
```

```
Output directory: /home/dka268/epic-ecal-sciglass/configurations/85da4603364a664c6b4bf8a3375e5896
Replacing existing path!
Output directory: /home/dka268/epic-ecal-sciglass/configurations/2174f7b867e834998e2363b1c962705c
Replacing existing path!
Output directory: /home/dka268/epic-ecal-sciglass/configurations/54d07c26f7cf2a94aaa3060ca4db88d2
Replacing existing path!
Output directory: /home/dka268/epic-ecal-sciglass/configurations/6efc7120f2773fdf2d922a63f5f679c5
Replacing existing path!
```




e⁻ (reference detector)
π⁻ (reference detector)
e⁻ (detector with tilt support)
π⁻ (detector with tilt support)
e⁻ (+5° tilt)
π⁻ (+5° tilt)
e⁻ (−5° tilt)
π⁻ (−5° tilt)
e⁻ (+10° tilt)
π⁻ (+10° tilt)
e⁻ (−10° tilt)
π⁻ (−10° tilt)
reference detector
detector with tilt support
+5° tilt
−5° tilt
+10° tilt
−10° tilt
|p_thrown| = 0.2 GeV
|p_thrown| = 0.5 GeV
|p_thrown| = 1.0 GeV
|p_thrown| = 2.0 GeV
|p_thrown| = 3.0 GeV
|p_thrown| = 5.0 GeV
|p_thrown| = 8.0 GeV
|p_thrown| = 10.0 GeV
|p_thrown| = 15.0 GeV
|p_thrown| = 18.0 GeV
Number of events
π⁻ rejection factor
Probability
Electron efficiency, %



6.1.8 Use of clustering

decode_sciglass_cellID_def

```python
def decode_sciglass_cellID(cellID):
    sector = (cellID >> np.int64(8)) & np.int64(0xFF)
    row = (cellID >> np.int64(16)) & np.int64(0xFF)
    tower = (cellID >> np.int64(24)) & np.int64(0xFF)
    # convert to a signed int without actually narrowing the bit width
    if isinstance(cellID, ak.Array):
        tower = ak.values_astype(tower, np.int64)
    tower = ((tower + np.int64(0x80)) % np.int64(0x100)) - np.int64(0x80)
    if isinstance(cellID, np.ndarray) or isinstance(cellID, ak.Array):
        _all = ak.all
    else:
        _all = lambda x: x
    assert _all(row >= 0)
    assert _all(row < 5)
    assert _all(sector >= 0)
    assert _all(sector < 24)
    return sector, row, tower

# check that scalar input works
np.testing.assert_equal(
    decode_sciglass_cellID(0),
    (0, 0, 0),
)
# check for numpy arrays
np.testing.assert_equal(
    decode_sciglass_cellID(np.array([1 << 8, 2 << 16, 3 << 24])),
    (
        [1, 0, 0],
        [0, 2, 0],
        [0, 0, 3],
    ),
)
# check for signed tower id's
np.testing.assert_equal(
    decode_sciglass_cellID(0xFF << 24),
    (0, 0, -1), # np.int8(np.int64(0xFF)) = -1
)
np.testing.assert_equal(
    decode_sciglass_cellID(0x80 << 24),
    (0, 0, -128), # np.int8(np.int64(0x80)) = -128
)
```

```
np.testing.assert_equal(
    np.array(decode_sciglass_cellID(ak.Array([0xFF << 24, 0x80 << 24]))),
    [[0, 0], [0, 0], [-1, -128]],
)
```

A simple cluster finder that allows to select up to one cluster per event:

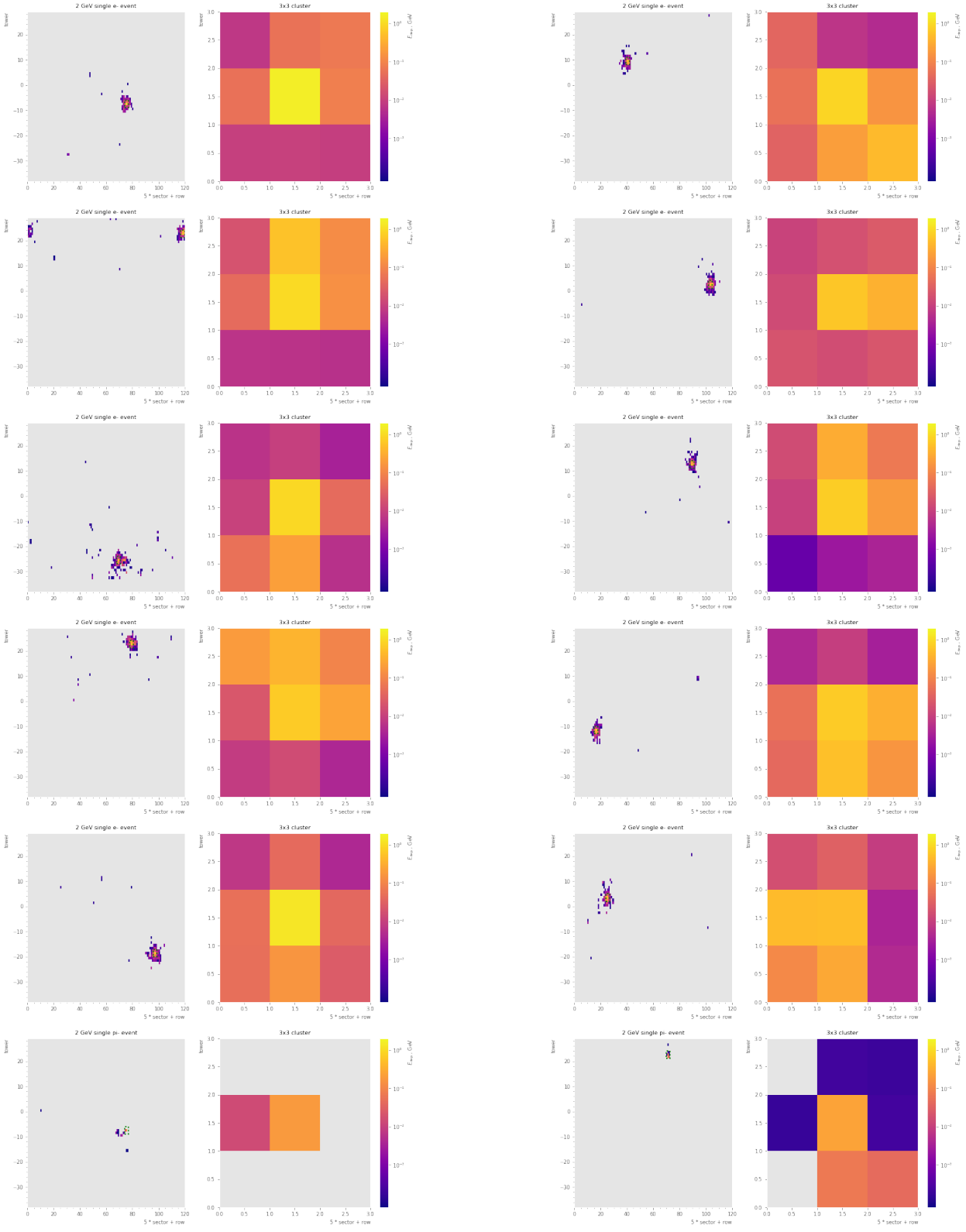
find_cluster_def

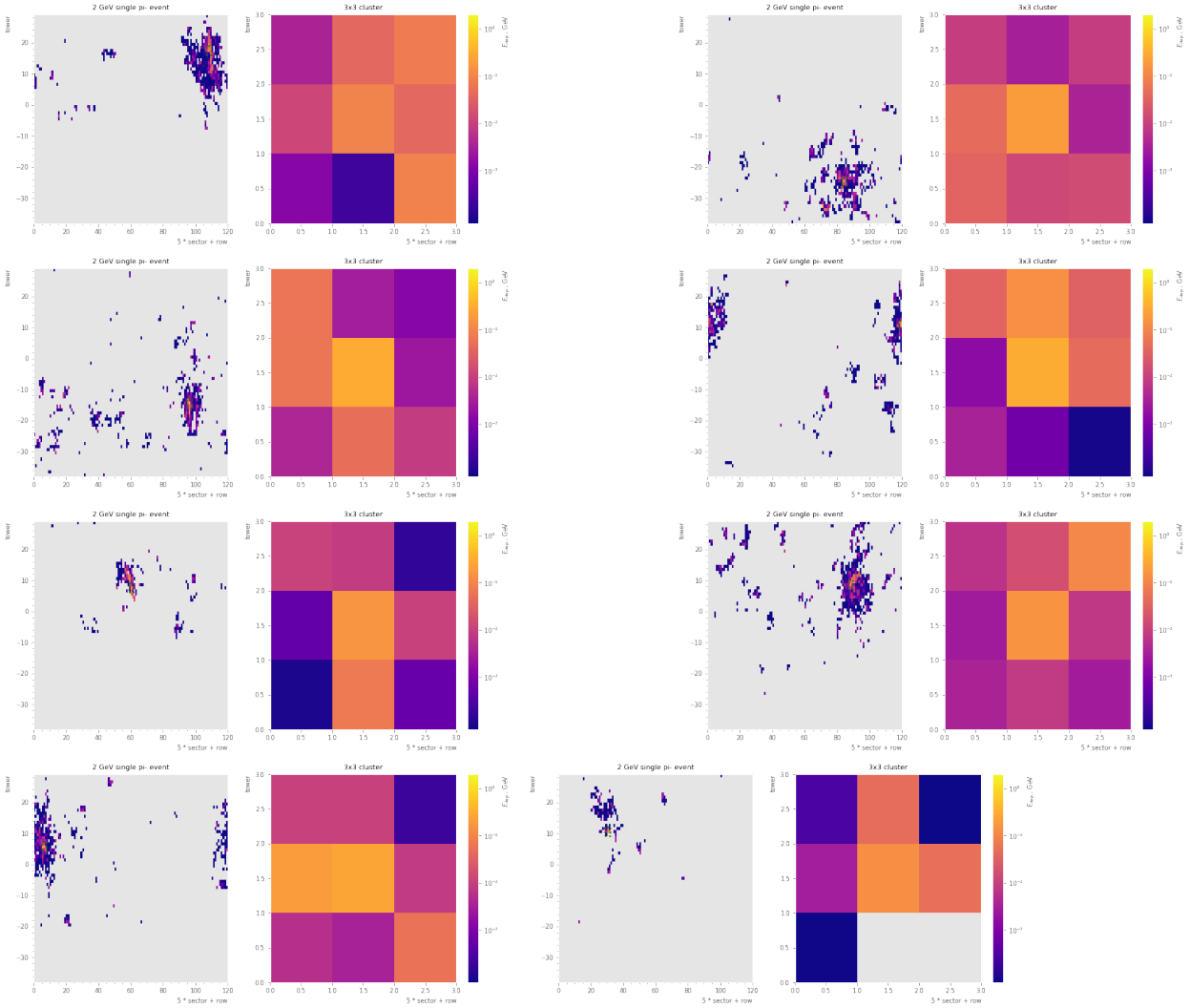
```
<<decode_sciglass_cellID_def>>

def find_cluster(events, size=3):
    X_MAX = NUM_ECALBARREL_SECTORS * NUM_ECALBARREL_ROWS

    edep = events["EcalBarrelSciGlassHits.energy"]
    sector, row, tower =
    ↪  decode_sciglass_cellID(events["EcalBarrelSciGlassHits.cellID"])

    x = sector * NUM_ECALBARREL_ROWS + row
    y = tower

    assert ak.all(x >= 0)
    assert ak.all(x < X_MAX)

    # Find index of a tower with peak energy
    ix = ak.argmax(edep, axis=1)

    # Broadcast index to enable use as Awkward-style index
    ix = ak.from_regular(ix[...,np.newaxis])

    # Center cluster at the peak energy tower
    x_clust = ak.firsts(x[ix])
    y_clust = ak.firsts(y[ix])

    egrid = np.moveaxis([[
        np.array(ak.fill_none(
            ak.sum(
                edep[(x == ((x_clust + xoff) % X_MAX)) & (y == y_clust + yoff)],
                axis=1
            ),
            0.0
        ))
        for yoff in range(-(size // 2), (size // 2) + 1)
    ]
        for xoff in range(-(size // 2), (size // 2) + 1)
    ], 2, 0)

    clusters = ak.from_iter(dict(
        ix = ix,
        x = x_clust,
        y = y_clust,
        sector = x_clust // NUM_ECALBARREL_ROWS,
        row = x_clust % NUM_ECALBARREL_ROWS,
        tower = y_clust,
        egrid = egrid,
```

```
    ))

    return clusters


_clusters = find_cluster(ak.Array([
    {
        "EcalBarrelSciGlassHits.energy" : [1, 2],
        "EcalBarrelSciGlassHits.cellID" : [1 << 16, 2 << 16],
    },
    {
        "EcalBarrelSciGlassHits.energy" : [1],
        "EcalBarrelSciGlassHits.cellID" : [0],
    },
    {
        "EcalBarrelSciGlassHits.energy" : [],
        "EcalBarrelSciGlassHits.cellID" : [],
    },
]))
np.testing.assert_equal(
    ak.fill_none(_clusters.ix, np.nan).to_numpy(),
    [[1], [0], [np.nan]]
)
np.testing.assert_equal(
    ak.fill_none(_clusters.sector, np.nan).to_numpy(),
    [0, 0, np.nan]
)
np.testing.assert_equal(
    ak.fill_none(_clusters.row, np.nan).to_numpy(),
    [2, 0, np.nan]
)
np.testing.assert_equal(
    ak.fill_none(_clusters.tower, np.nan).to_numpy(),
    [0, 0, np.nan]
)
np.testing.assert_equal(
    _clusters.egrid.to_numpy(),
    [
        [[0., 1., 0.],
         [0., 2., 0.],
         [0., 0., 0.]],
        [[0., 0., 0.],
         [0., 1., 0.],
         [0., 0., 0.]],
        [[0., 0., 0.],
         [0., 0., 0.],
         [0., 0., 0.]],
    ],
)
```

```
<<decode_sciglass_cellID_def>>
<<find_cluster_def>>
```

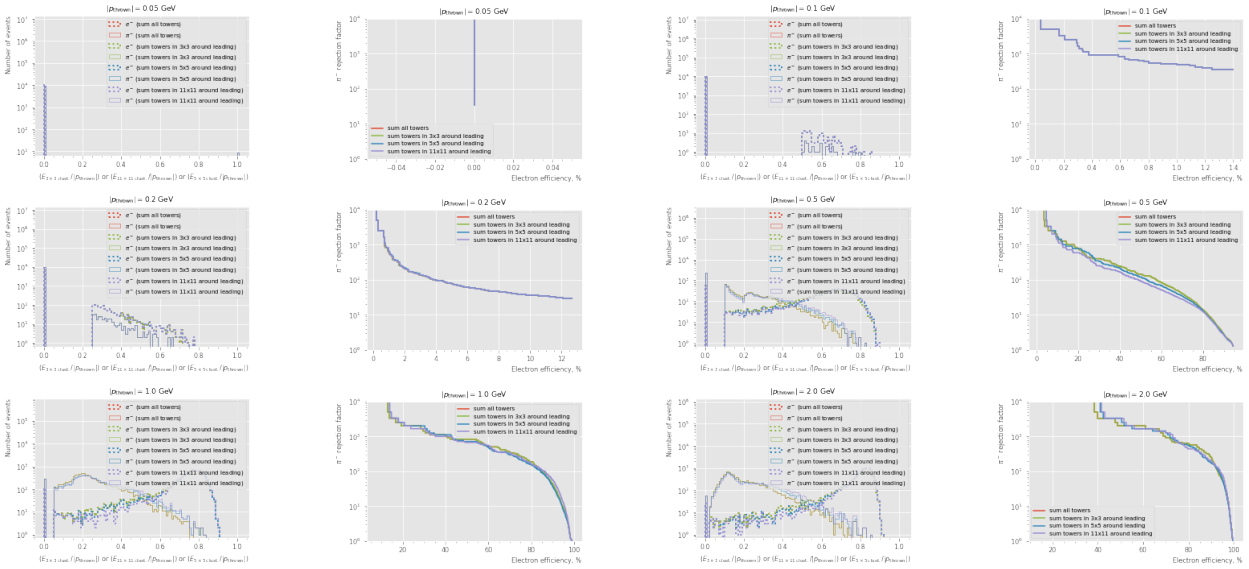
```
output_dir = Path("results/find_cluster")
output_dir.mkdir(parents=True, exist_ok=True)

p_thrown = 2 # GeV
for particle in ["e-", "pi-"]:
    simu = client.compute(run_ddsim(particle=particle, p_min=p_thrown, p_max=p_thrown,
    ↪  **dict(common_sim_params, num_events=10))).result()

    sector, row, tower = decode_sciglass_cellID(simu["EcalBarrelSciGlassHits.cellID"])

    for event_id in range(10):
        x = (sector * 5 + row)[event_id].to_numpy()
        y = (tower)[event_id].to_numpy()
        edep = simu["EcalBarrelSciGlassHits.energy"][event_id].to_numpy()

        size = 3
        clusters = find_cluster(simu, size=size)

        bins = (
            np.linspace(0, 24 * 5, 24 * 5 + 1),
            np.linspace(-38, 28 + 1, 38 + 28 + 1 + 1),
        )

        _, ax = plt.subplots(ncols=2, figsize=(11, 5),
        ↪  gridspec_kw=dict(width_ratios=(5,6)))

        hist_vis = dict(
            cmap="plasma", norm=mpl.colors.SymLogNorm(linthresh=1 * MeV, vmin=1e-10,
            ↪  vmax=p_thrown),
        )

        plt.sca(ax[0])
        _, _, _, m = plt.hist2d(x, y, weights=edep, bins=bins, cmin=1e-10, **hist_vis)
        if ak.sum(edep) != 0:
            plt.gca().add_patch(mpl.patches.Rectangle((clusters["x"][event_id] - size
            ↪  // 2, clusters["y"][event_id] - size // 2), size, size, fill=False,
            ↪  edgecolor="green", ls=":", lw=2))
        plt.title(f"{p_thrown} GeV single {particle} event")
        plt.xlabel("5 * sector + row", loc="right")
        plt.ylabel("tower", loc="top")
        plt.minorticks_on()

        plt.sca(ax[1])
        plt.pcolormesh(np.ma.masked_values(clusters["egrid"][event_id].to_numpy().T,
        ↪  0.), shading="flat", **hist_vis)
        plt.colorbar(m, ax=ax[1]).set_label(r"$E_{\mathrm{dep.}}$, GeV", loc="top")
        plt.title(f"{size}x{size} cluster")
        plt.xlabel("5 * sector + row", loc="right")
        plt.ylabel("tower", loc="top")

        plt.savefig(output_dir / f"event_{particle}_{p_thrown:.2f}_{event_id}.pdf")
        plt.show()
```

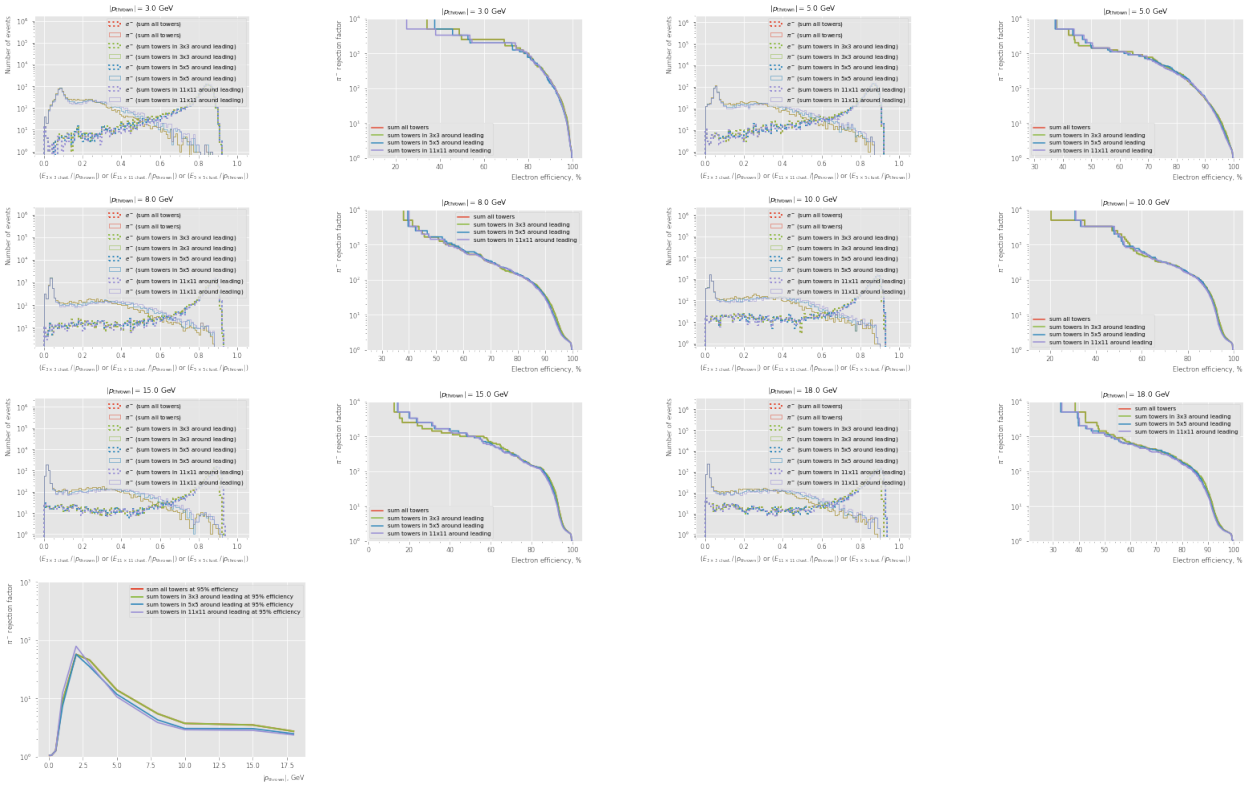

2 GeV single e- event
3x3 cluster
tower
5 * sector + row
2 GeV single pi- event




```
<<sim_settings>>

sim_settings = [
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
        ),
        label="sum all towers",
        visual=dict(
            color="C0",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
```

```
        ),
        label="sum towers in 3x3 around leading",
        visual=dict(
            color="C5",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=5), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
        ),
        label="sum towers in 5x5 around leading",
        visual=dict(
            color="C1",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=11), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
        ),
        label="sum towers in 11x11 around leading",
        visual=dict(
            color="C2",
        ),
    ),
]

output_dir = Path("results/find_cluster")

<<pion_rej_vs_pt>>
```

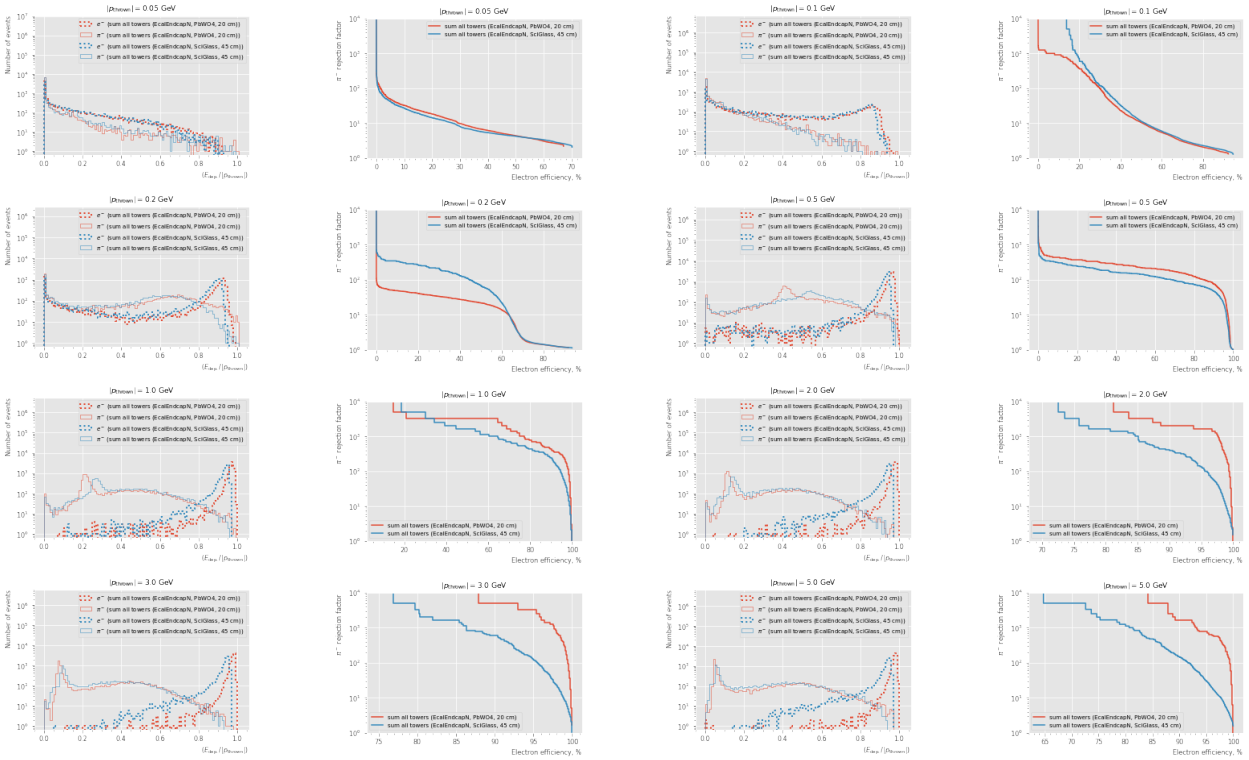


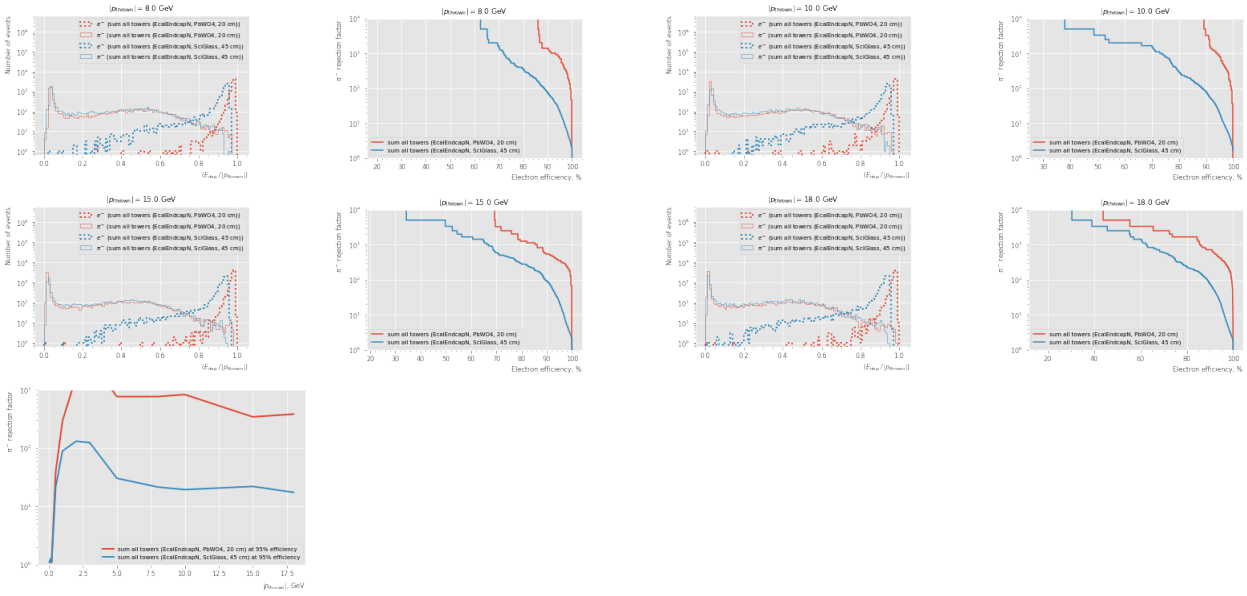


6.1.9 Negative Endcap

```
endcapn_sim_params = dict(
    num_events=NUM_EVENTS_DEFAULT,
    theta_min=2.82,
    theta_max=3.08,
    branches = [
        "EcalEndcapNHits.cellID",
        "EcalEndcapNHits.energy",
        "MCParticles.momentum.x",
        "MCParticles.momentum.y",
        "MCParticles.momentum.z",
    ],
)

def endcapn_edep_sum_classifier(events):
    p_thrown = np.sqrt(
        events["MCParticles.momentum.x"][:,0] ** 2
        + events["MCParticles.momentum.y"][:,0] ** 2
        + events["MCParticles.momentum.z"][:,0] ** 2
    )
    return ak.sum(events["EcalEndcapNHits.energy"], axis=1) /  p_thrown

endcapn_edep_sum_classifier.xlabel = edep_sum_classifier.xlabel


sim_settings = [
```

```
    dict(
        classifier=endcapn_edep_sum_classifier,
        sim=dict(
            endcapn_sim_params,
        ),
        label="sum all towers (EcalEndcapN, PbWO4, 20 cm)",
        visual=dict(
            color="C0",
        ),
    ),
    dict(
        classifier=endcapn_edep_sum_classifier,
        sim=dict(
            endcapn_sim_params,
            detector_path=endcapn_sciglass,
        ),
        label="sum all towers (EcalEndcapN, SciGlass, 45 cm)",
        visual=dict(
            color="C1",
        ),
    ),
]

output_dir = Path("results/endcap")

<<pion_rej_vs_pt>>
```

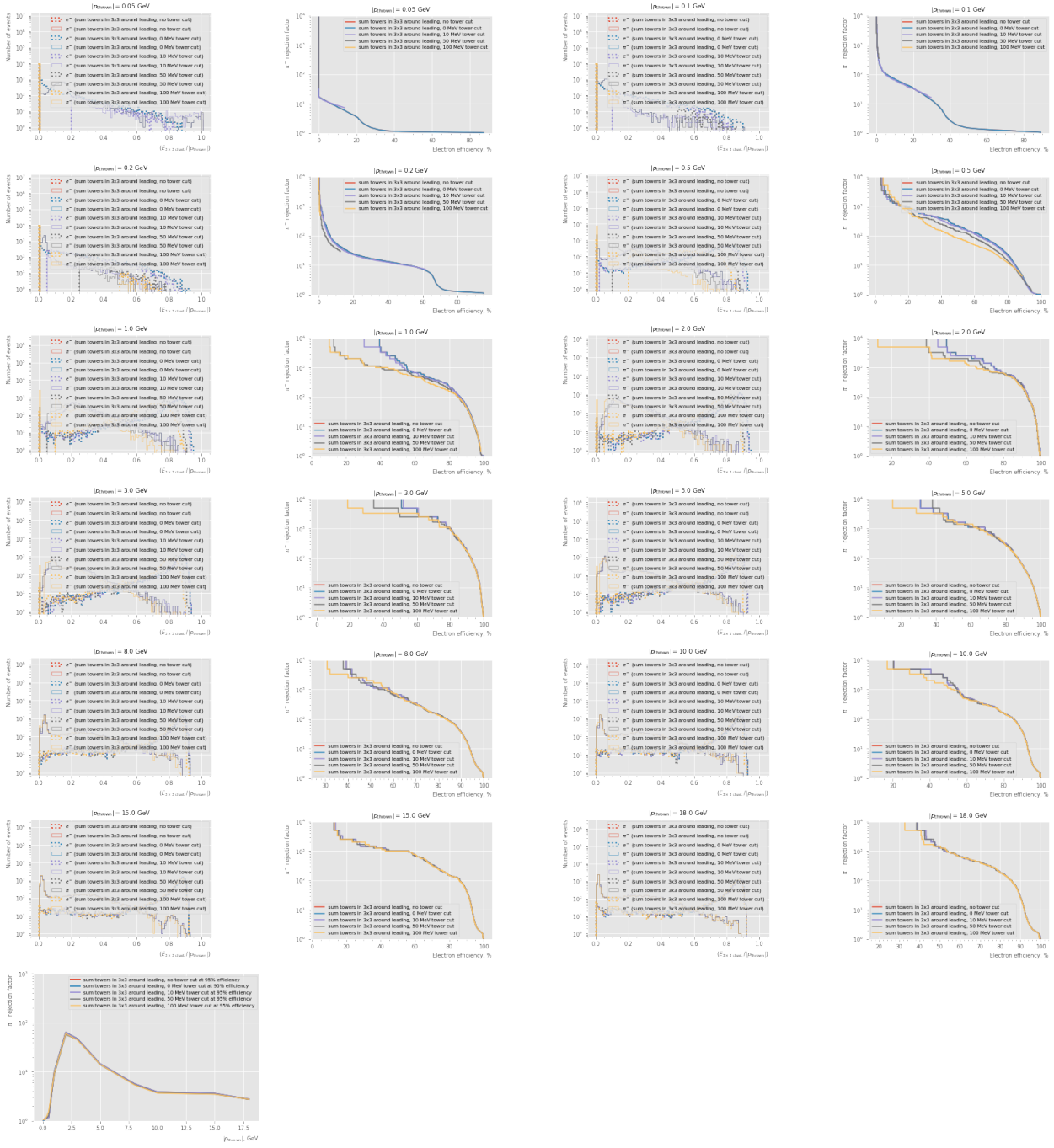




6.1.10 Pion rejection vs energy threshold

```
<<sim_settings>>

thresholds = [0, 10 * MeV, 50 * MeV, 100 * MeV]

sim_settings = [
    dict(
        classifier=edep_cluster_sum_classifier(size=3),
        sim=dict(
            common_sim_params,
        ),
        label="sum towers in 3x3 around leading, no tower cut",
        visual=dict(
            color="C0",
        ),
    ),
] + [
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3),
        ↪  threshold),
        sim=dict(
            common_sim_params,
        ),
        label=fr"sum towers in 3x3 around leading, {threshold / MeV:.0f} MeV tower
        ↪  cut",
        visual=dict(
            color=f"C{ix + 1}",
        ),
    ) for ix, threshold in enumerate(thresholds)
]

output_dir = Path("results/tower_energy_cut")
```

```
<<pion_rej_vs_pt>>
```



6.1.11 Machine learning

We start by defining a basic setting to be used to run a simulation to produce a training dataset. The classifier is set to use the same 3x3 tower energy values:

ml_sim_settings

```
<<sim_settings>>

sim_settings = [
    dict(
        classifier=enable_energy_threshold(edep_cluster_sum_classifier(size=3), 50 *
        ↪  MeV),
        sim=dict(
            common_sim_params,
            seed=31137, # training seed
        ),
        label="sum towers in 3x3 around leading",
        visual=dict(
            color="C0",
        ),
    ),
]
```

The following functions define sets of features to be used for Machine Learning. A simple feature set is obtained by taking 9 energy values for the 3x3 clusters:

to_x_def

```
<<find_cluster_def>>

def to_x(events):
    clusters = find_cluster(events)
    return np.array(ak.flatten(clusters.egrid, axis=-1))
```

A more sophisticated particle ID could rely on the information from tracking. Since this relies on unreconstructed simulation, let's rely on the truth momentum to investigate the best-possible performance.

to_x_with_3momentum_def

```
def momentum(events):
    px = ak.firsts(events["MCParticles.momentum.x"]).to_numpy()
    py = ak.firsts(events["MCParticles.momentum.y"]).to_numpy()
    pz = ak.firsts(events["MCParticles.momentum.z"]).to_numpy()
    return np.sqrt(px**2 + py**2 + pz**2)


def theta(events):
    px = ak.firsts(events["MCParticles.momentum.x"]).to_numpy()
    py = ak.firsts(events["MCParticles.momentum.y"]).to_numpy()
    pz = ak.firsts(events["MCParticles.momentum.z"]).to_numpy()
    return np.arctan2(np.sqrt(px**2 + py**2), pz)


def phi(events):
    px = ak.firsts(events["MCParticles.momentum.x"]).to_numpy()
    py = ak.firsts(events["MCParticles.momentum.y"]).to_numpy()
    return np.arctan2(py, px)


<<to_x_def>>
```

```
def to_x_with_momentum(events):
    return np.concatenate([
        to_x(events) / momentum(events)[...,np.newaxis],
        np.log(momentum(events)[...,np.newaxis]),
    ], axis=1)


def to_x_with_3momentum(events):
    return np.concatenate([
        to_x(events) / momentum(events)[...,np.newaxis],
        np.log(momentum(events))[...,np.newaxis],
        theta(events)[...,np.newaxis],
        np.mod(phi(events), 2 * np.pi / NUM_ECALBARREL_SECTORS)[...,np.newaxis],
    ], axis=1)


def to_x_with_3momentum_clu_pos(events):
    clusters = find_cluster(events)
    cluster_phi_offset = np.mod(phi(events) + np.pi, 2 * np.pi) / DELTA_PHI_PER_ROW -
    ↪  (np.mod(clusters.sector.to_numpy() + NUM_ECALBARREL_SECTORS / 2,
    ↪  NUM_ECALBARREL_SECTORS) * 5 + clusters.row.to_numpy())
    cluster_eta_offset = theta2eta(theta(events)) - clusters.tower.to_numpy() *
    ↪  np.mean(theta2eta(theta(events)) / clusters.tower.to_numpy())
    return np.concatenate([
        to_x(events) / momentum(events)[...,np.newaxis],
        np.log(momentum(events))[...,np.newaxis],
        theta(events)[...,np.newaxis],
        np.mod(phi(events), 2 * np.pi / NUM_ECALBARREL_SECTORS)[...,np.newaxis],
        clusters.row.to_numpy()[...,np.newaxis],
        clusters.tower.to_numpy()[...,np.newaxis],
        cluster_phi_offset[...,np.newaxis],
        cluster_eta_offset[...,np.newaxis],
    ], axis=1)
```

The φ angle is taken modulo the anlge ot the axial symmetry of the calorimeter. This is done to take away from ML the burden of learning this property.

train_def

```
@dask.delayed(pure=True)
def do_fit(clf, x_train, y_train):
    with joblib.parallel_backend("dask"):
        return clf.fit(x_train, y_train)


def train(electrons_train, pions_train, to_x, clf):
    electrons_train = client.gather(electrons_train)
    pions_train = client.gather(pions_train)
    if isinstance(electrons_train, list):
        electrons_train = ak.concatenate(electrons_train, axis=0)
    if isinstance(pions_train, list):
        pions_train = ak.concatenate(pions_train, axis=0)

    x_train = np.concatenate([
        to_x(electrons_train),
```

```
        to_x(pions_train),
    ])

    y_train = np.concatenate([
        np.zeros(len(electrons_train)),
        np.ones(len(pions_train)),
    ])

    rng = np.random.default_rng(42)
    ixs, = np.indices(y_train.shape[:1])
    rng.shuffle(ixs)
    x_train = x_train[ixs]
    y_train = y_train[ixs]

    clf_comp = client.compute(do_fit(clf, x_train, y_train))

    def classifier(events):
        clf = clf_comp.result()
        return clf.predict_proba(to_x(events))[:,0]
    classifier.unwrapped = clf_comp
    classifier.xlabel = "Probability"

    return classifier
```

train

```
<<ml_sim_settings>>

particles = ["e-", "pi-"]
output_dir = Path("results/detector_variants_ml")

sim = {}
for p_thrown in energies:
    <<book_sim>>

sim_train = sim
sim = {}

from sklearn.neural_network import MLPClassifier
from xgboost import XGBClassifier

<<train_def>>
<<to_x_def>>
<<to_x_with_3momentum_def>>
<<energy_dispatch_def>>

mlp_classifier = energy_dispatch({
    np.rint(p_thrown * 1e3): train(
        sim_train[p_thrown][0]["e-"],
        sim_train[p_thrown][0]["pi-"],
        enable_energy_threshold(to_x, 50 * MeV),
        MLPClassifier(learning_rate="constant", hidden_layer_sizes=(1000,),
        ↪  max_iter=1000, random_state=1, early_stopping=True, verbose=False),
    )
```

```
    for p_thrown in energies
})
mlp_with_p_classifier = train(
    [sim_train[p_thrown][0]["e-"] for p_thrown in energies],
    [sim_train[p_thrown][0]["pi-"] for p_thrown in energies],
    enable_energy_threshold(to_x_with_momentum, 50 * MeV),
    MLPClassifier(learning_rate="constant", hidden_layer_sizes=(1000,), max_iter=1000,
    ↪  random_state=1, early_stopping=True, verbose=False),
)
mlp_with_p_vec_classifier = train(
    [sim_train[p_thrown][0]["e-"] for p_thrown in energies],
    [sim_train[p_thrown][0]["pi-"] for p_thrown in energies],
    enable_energy_threshold(to_x_with_3momentum, 50 * MeV),
    MLPClassifier(learning_rate="constant", hidden_layer_sizes=(1000,), max_iter=1000,
    ↪  random_state=1, early_stopping=True, verbose=False),
)

xgboost_classifier = energy_dispatch({
    np.rint(p_thrown * 1e3): train(
        sim_train[p_thrown][0]["e-"],
        sim_train[p_thrown][0]["pi-"],
        enable_energy_threshold(to_x, 50 * MeV),
        XGBClassifier(),
    )
    for p_thrown in energies
})
xgboost_with_p_classifier = train(
    [sim_train[p_thrown][0]["e-"] for p_thrown in energies],
    [sim_train[p_thrown][0]["pi-"] for p_thrown in energies],
    enable_energy_threshold(to_x_with_momentum, 50 * MeV),
    XGBClassifier(),
)
xgboost_with_p_vec_classifier = train(
    [sim_train[p_thrown][0]["e-"] for p_thrown in energies],
    [sim_train[p_thrown][0]["pi-"] for p_thrown in energies],
    enable_energy_threshold(to_x_with_3momentum, 50 * MeV),
    XGBClassifier(),
)
```

At this point there are two kinds of Multi-layer Perceptron classifiers: one set of classifiers trained for each specific momentum, and one trained on all available momenta at once. In order to benchmark the earlier one, the `energy_dispatch` wrapper is used to pick the right classifier based on the momentum:

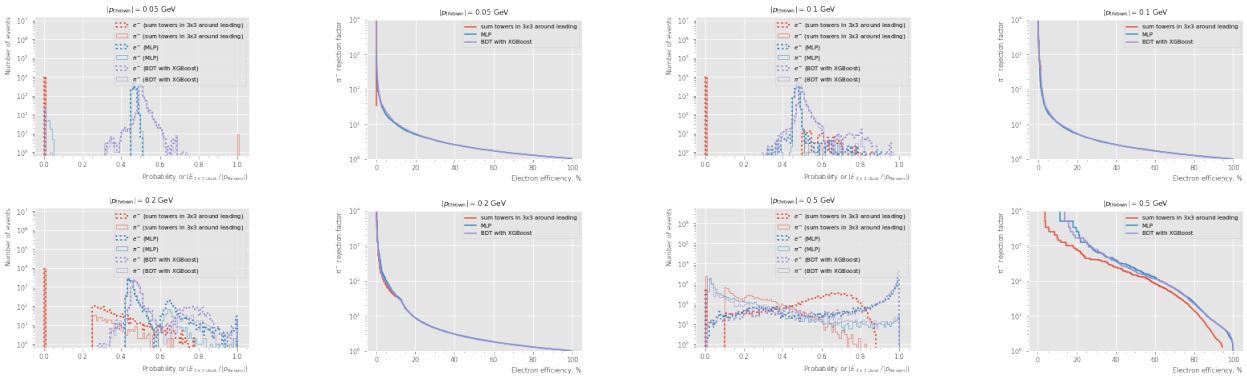
energy_dispatch_def

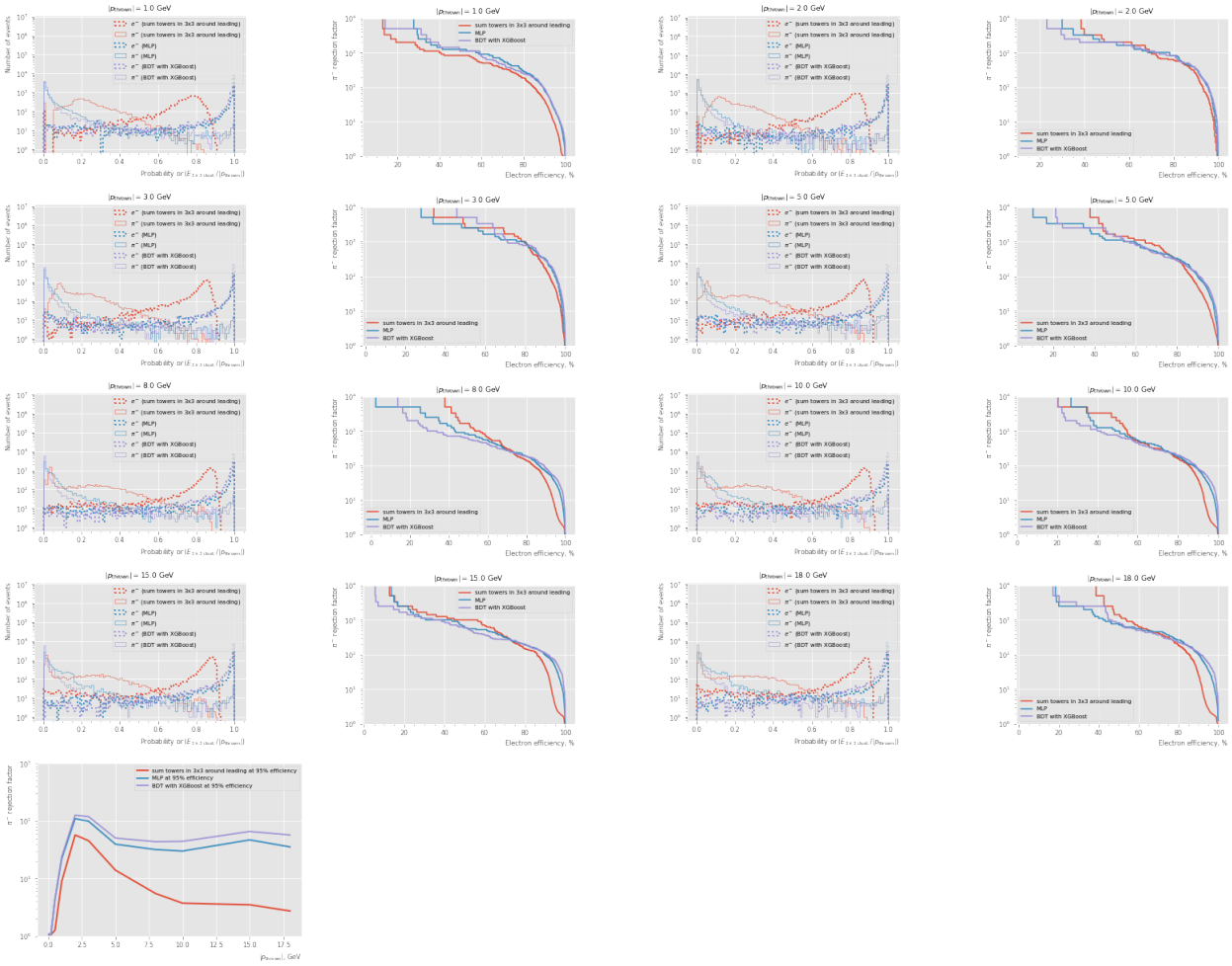
```
def energy_dispatch(classifiers):
    def classifier(events):
        p_thrown = np.mean(np.sqrt(
            events["MCParticles.momentum.x"][:,0] ** 2
            + events["MCParticles.momentum.y"][:,0] ** 2
            + events["MCParticles.momentum.z"][:,0] ** 2
        ).to_numpy())
        return classifiers[np.rint(p_thrown * 1e3)](events)
    xlabels = set([c.xlabel for c in classifiers.values() if hasattr(c, "xlabel")])
    if xlabels:
```

```
        classifier.xlabel = " or ".join(xlabels)
    return classifier
```

Note that this implementation doesn't support mixed values.
Now is the time to evaluate the resulting ML classifiers on the our dataset:

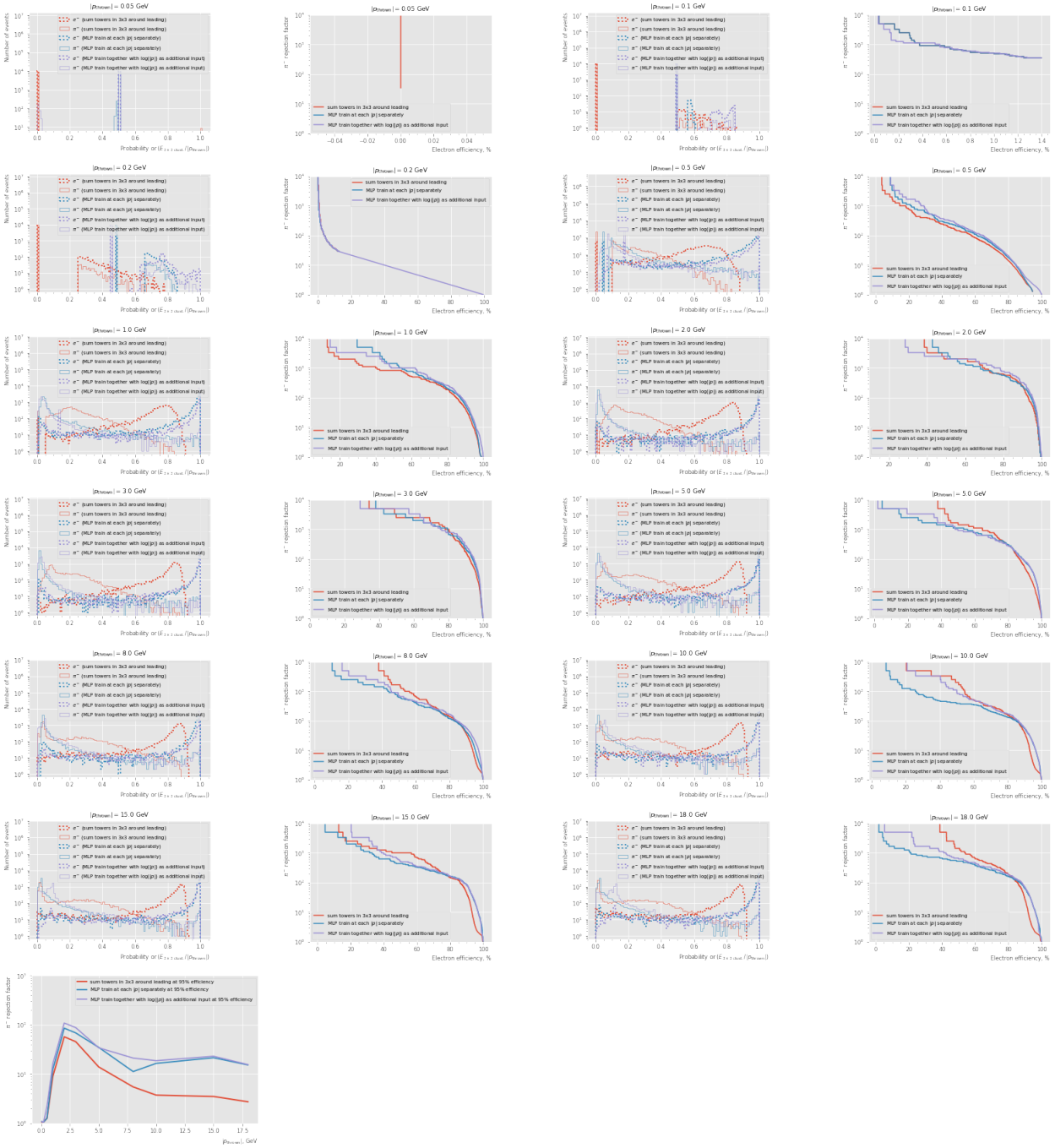
```
<<ml_sim_settings>>

sim_settings[0]["sim"]["seed"] = 1 # evaluation seed
sim_settings += [
    dict(
        classifier=enable_energy_threshold(mlp_with_p_vec_classifier, 50 * MeV),
        sim=dict(
            common_sim_params,
        ),
        label="MLP",
        visual=dict(
            color="C1",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(xgboost_with_p_vec_classifier, 50 * MeV),
        sim=dict(
            common_sim_params,
        ),
        label="BDT with XGBoost",
        visual=dict(
            color="C2",
        ),
    ),
]

output_dir = Path("results/mlp_vs_xgboost")

<<pion_rej_vs_pt>>
```



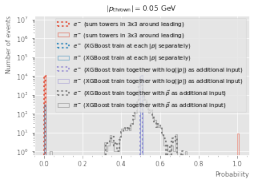
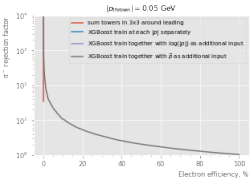
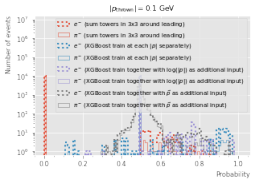
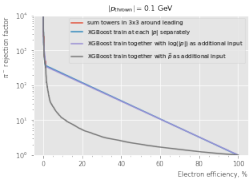


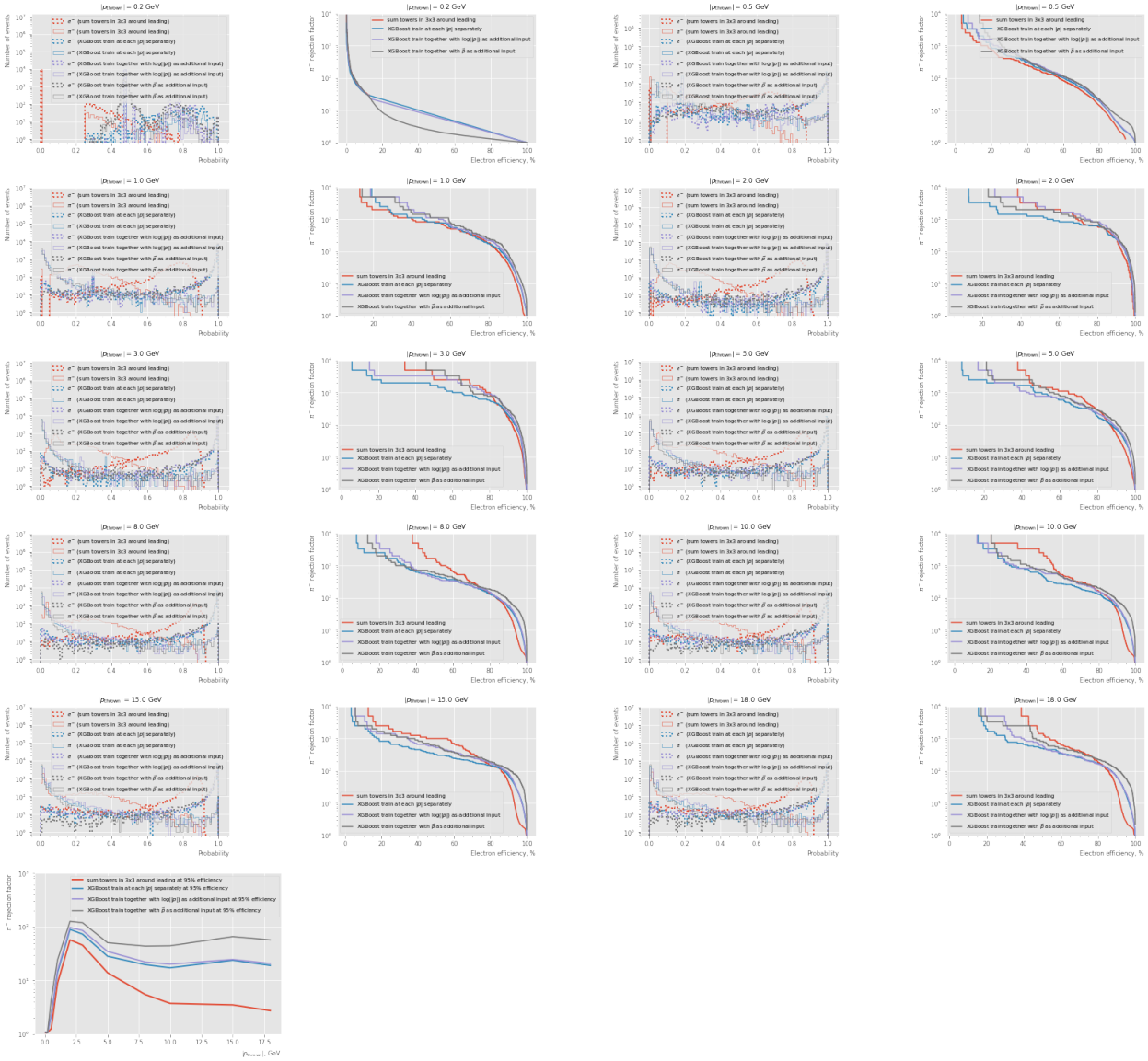
```
<<ml_sim_settings>>

sim_settings[0]["sim"]["seed"] = 1 # evaluation seed
sim_settings += [
    dict(
        classifier=enable_energy_threshold(mlp_classifier, 50 * MeV),
        sim=dict(
            common_sim_params,
        ),
        label="MLP train at each $|p|$ separately",
        visual=dict(
            color="C1",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(mlp_with_p_classifier, 50 * MeV),
        sim=dict(
            common_sim_params,
        ),
        label=r"MLP train together with $\mathrm{log}(|p|)$ as additional input",
        visual=dict(
            color="C2",
```

```
        ),
    ),
]

output_dir = Path("results/mlp_input_variations")

<<pion_rej_vs_pt>>
```



```
<<ml_sim_settings>>

sim_settings[0]["sim"]["seed"] = 1 # evaluation seed
sim_settings += [
    dict(
        classifier=enable_energy_threshold(xgboost_classifier, 50 * MeV),
        sim=dict(
            common_sim_params,
        ),
        label="XGBoost train at each $|p|$ separately",
        visual=dict(
            color="C1",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(xgboost_with_p_classifier, 50 * MeV),
        sim=dict(
            common_sim_params,
        ),
        label=r"XGBoost train together with $\mathrm{log}(|p|)$ as additional input",
        visual=dict(
            color="C2",
        ),
    ),
    dict(
        classifier=enable_energy_threshold(xgboost_with_p_vec_classifier, 50 * MeV),
        sim=dict(
            common_sim_params,
        ),
        label=r"XGBoost train together with $\vec{p}$ as additional input",
        visual=dict(
            color="C3",
        ),
    ),
]

for setting in sim_settings:
    setting["classifier"].xlabel = "Probability"

output_dir = Path("results/xgboost_input_variations")

<<pion_rej_vs_pt>>
```

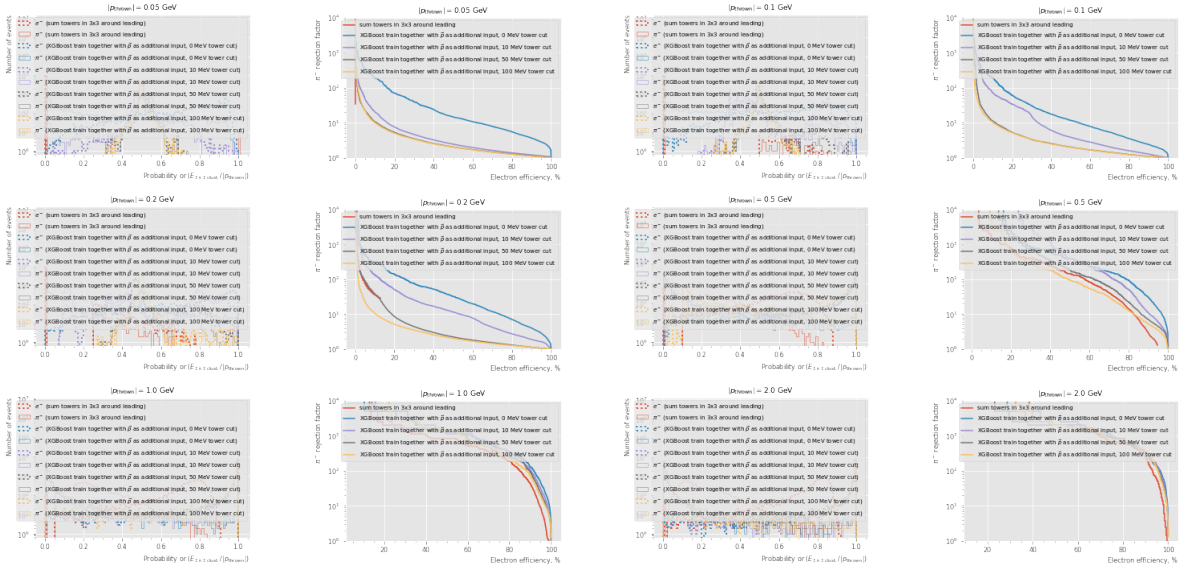





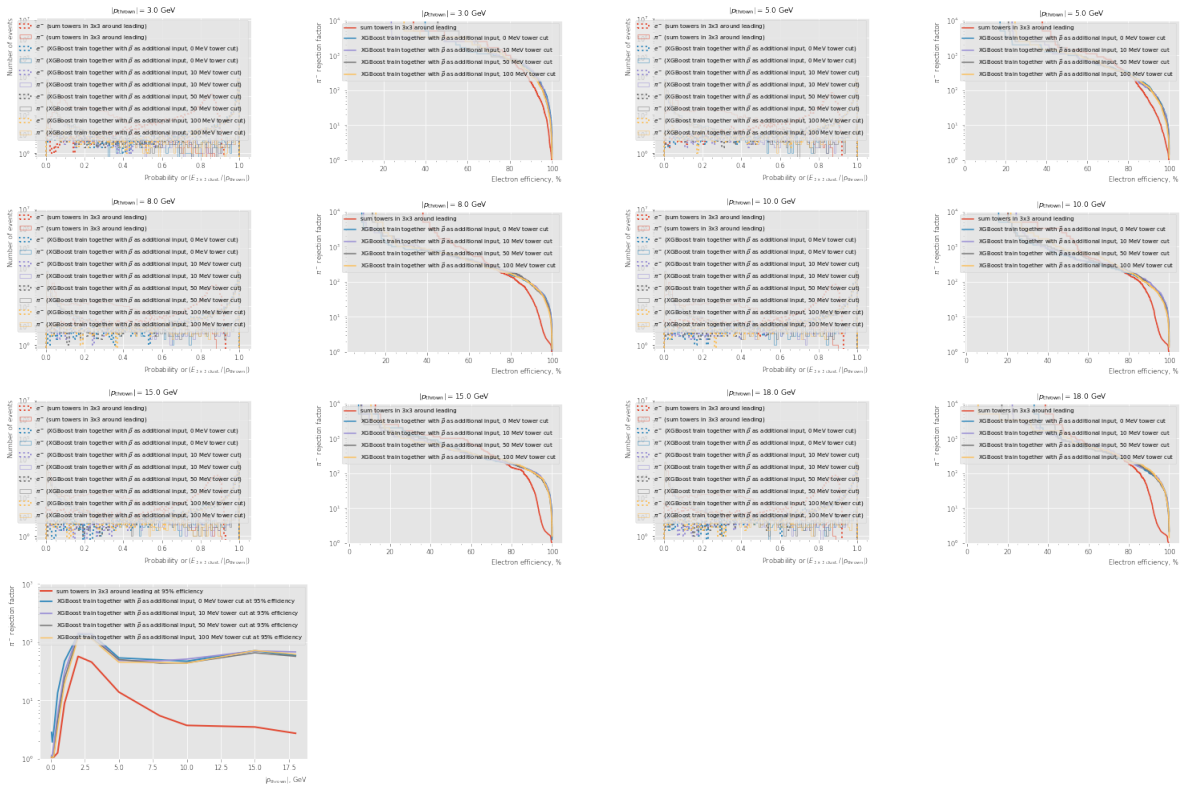


1. Pion rejection vs energy threshold for ML

```
<<ml_sim_settings>>

particles = ["e-", "pi-"]

sim = {}
for p_thrown in energies:
    <<book_sim>>

sim_train = sim
sim = {}

from xgboost import XGBClassifier

<<train_def>>
<<to_x_with_3momentum_def>>
```

```
thresholds = [0, 10 * MeV, 50 * MeV, 100 * MeV]

xgboost_with_p_vec_classifier_threshold = [
    train(
        [sim_train[p_thrown][0]["e-"] for p_thrown in energies],
        [sim_train[p_thrown][0]["pi-"] for p_thrown in energies],
        enable_energy_threshold(to_x_with_3momentum, threshold),
        XGBClassifier(),
    ) for threshold in thresholds
]

sim_settings[0]["sim"]["seed"] = 1 # evaluation seed
sim_settings += [
    dict(
        classifier=enable_energy_threshold(classifier, threshold),
        sim=dict(
            common_sim_params,
        ),
        label=fr"XGBoost train together with $\vec{{p}}$ as additional input,
        ↪  {threshold / MeV:.0f} MeV tower cut",
        visual=dict(
            color=f"C{ix + 1}",
        ),
    ) for ix, (threshold, classifier) in enumerate(zip(thresholds,
    ↪  xgboost_with_p_vec_classifier_threshold))
]

output_dir = Path("results/tower_energy_cut_ml")

<<pion_rej_vs_pt>>
```





6.2 Island Clustering

find_island_cluster_def

```
<<decode_sciglass_cellID_def>>


def _find_island_cluster(events, debug=False):
    sector, row, tower =
    ↪  decode_sciglass_cellID(events["EcalBarrelSciGlassHits.cellID"])

    result = []

    builder = ak.ArrayBuilder()

    for event_id in range(len(events)):
        x = (sector * 5 + row)[event_id].to_numpy().astype(int)
        y = (tower)[event_id].to_numpy().astype(int)
        edep = events["EcalBarrelSciGlassHits.energy"][event_id].to_numpy()
        hit_x = events["EcalBarrelSciGlassHits.position.x"][event_id].to_numpy()
        hit_y = events["EcalBarrelSciGlassHits.position.y"][event_id].to_numpy()
        hit_z = events["EcalBarrelSciGlassHits.position.z"][event_id].to_numpy()

        if debug:
            bin_padding = 10 # fit "overhanging" clusters
            bins = (
                np.linspace(-bin_padding, 24 * 5 + bin_padding, 24 * 5 + 2 *
                ↪  bin_padding + 1),
                np.linspace(-38, 28 + 1, 38 + 28 + 1 + 1),
```

```
            )
            _, ax = plt.subplots(ncols=3, figsize=(16, 5),
            ↪  gridspec_kw=dict(width_ratios=(6,7,6)))
            hist_vis = dict(
                cmin=1e-10, cmap="plasma", norm=mpl.colors.SymLogNorm(linthresh=1 *
                ↪  MeV, vmin=1e-10, vmax=p_thrown),
            )
            plt.sca(ax[0])
            _, _, _, m = plt.hist2d(x, y, weights=edep, bins=bins, **hist_vis)
            plt.xlim(0, 24 * 5)

            plt.sca(ax[1])
            ixs = np.argsort(edep)[-3:]
            x_max = x[ixs[0]]
            y_max = int(np.mean(y[ixs]))
            wraparound_x = lambda x: np.mod(x - int(x_max) + 24 * 5 // 2, 24 * 5) +
            ↪  int(x_max) - 24 * 5 // 2
            _, _, _, m = plt.hist2d(wraparound_x(x), y, weights=edep, bins=bins,
            ↪  **hist_vis)
            x_max = int(np.mean(wraparound_x(x[ixs])))
            plt.xlim(x_max - 5.5, x_max + 5.5)
            plt.ylim(y_max - 5.5, y_max + 5.5)
            plt.colorbar().set_label(r"$E_{\mathrm{dep.}}$, GeV", labelpad=-1.,
            ↪  loc="top")

            plt.sca(ax[0])
            for _ax in ax[:2]:
                plt.sca(_ax)
                plt.xlabel("5 * sector + row", loc="right")
                plt.ylabel("tower", loc="top")
                plt.minorticks_on()

        adjacency_matrix = (
            np.min([
                abs(x[:,np.newaxis] - x[np.newaxis,:]),
                24 * 5 - abs(x[:,np.newaxis] - x[np.newaxis,:]),
            ], axis=0)
            +
            abs(y[:,np.newaxis] - y[np.newaxis,:])
        ) == 1

        from scipy.sparse.csgraph import connected_components
        num_candidates, tower_labels = connected_components(csgraph=adjacency_matrix,
        ↪  directed=False, return_labels=True)
        good_clusters = []
        for label in range(num_candidates):
            if np.max(edep[tower_labels == label]) > 50 * MeV:
                good_clusters.append(label)

        local_maxima = {}

        if debug:
            subcluster_ix = 0
```

```
        builder.begin_list()
        for cluster_ix, label in enumerate(good_clusters):
            builder.begin_record("Momentum4D")

            energy = np.sum(edep[tower_labels == label])
            d_x = np.sum(hit_x[tower_labels == label])
            d_y = np.sum(hit_y[tower_labels == label])
            d_z = np.sum(hit_z[tower_labels == label])
            norm = np.sqrt(d_x ** 2 + d_y ** 2 + d_z ** 2)

            builder.field("E")
            builder.real(energy)
            builder.field("px")
            builder.real(energy * d_x / norm)
            builder.field("py")
            builder.real(energy * d_y / norm)
            builder.field("pz")
            builder.real(energy * d_z / norm)

            neighbour_max_edep = np.array([
                np.max(edep[np.where(adjacent_mask)]) if np.any(adjacent_mask) else 0.
                for adjacent_mask in adjacency_matrix[tower_labels == label]
            ])
            is_local_maximum = neighbour_max_edep < edep[tower_labels == label]
            is_local_maximum &= edep[tower_labels == label] > 100 * MeV
            if debug: print(edep[tower_labels == label][is_local_maximum])

            distance = np.sqrt(
                np.min([
                    np.abs(x[:,np.newaxis] - x[np.newaxis,:]),
                    24 * 5 - abs(x[:,np.newaxis] - x[np.newaxis,:]),
                ], axis=0) ** 2
                +
                (y[:,np.newaxis] - y[np.newaxis,:]) ** 2
            )

            def f(lambda_):
                expected_edep = np.exp(
                    - distance[tower_labels == label,:][:,tower_labels ==
                    ↪  label][is_local_maximum] / lambda_
                ) * edep[tower_labels == label][is_local_maximum,np.newaxis]
                xi2 = np.sum(
                    (expected_edep.sum(axis=0) - edep[tower_labels == label]) ** 2
                ) / (p_thrown ** 2)
                return xi2

            if True:
                lams = np.geomspace(0.01, 10, 100)
                xi2s = np.vectorize(f)(lams)
                lambda_ = lams[np.argmin(xi2s)]
            else:
                import scipy.optimize
```

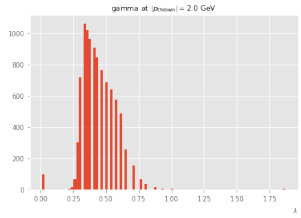
```
            res = scipy.optimize.minimize(f, 1., bounds=[(0.01, 10)])
            if res.success:
                lambda_ = res.x[0]
            else:
                lambda_ = 1.

        builder.field("lambda")
        builder.real(lambda_)

        if debug:
            plt.sca(ax[2])
            plt.plot(lams, xi2s)
            plt.xscale("log")
            plt.yscale("log")
            plt.title("Shower profile parameter scan")
            plt.ylabel(r"$X^2 = \frac{1}{(p_{\mathrm{thrown}})^2} \sum_h
            ↪  \left(\sum_m E_m \exp\left(-\frac{d_{hm}}{\lambda}\right) -
            ↪  E_h\right)^2$", labelpad=-8., loc="top")
            plt.xlabel(r"$\lambda$", loc="right")

        expected_edep = np.exp(
            - distance[tower_labels == label,:][:,tower_labels ==
            ↪  label][is_local_maximum] / lambda_
        ) * edep[tower_labels == label][is_local_maximum,np.newaxis]
        weights = expected_edep / expected_edep.sum(axis=0)[np.newaxis,:]

        builder.field("subclusters")
        builder.begin_list()
        for w in weights:
            builder.begin_record("Momentum4D")

            energy = w.dot(edep[tower_labels == label])
            d_x = w.dot(hit_x[tower_labels == label])
            d_y = w.dot(hit_y[tower_labels == label])
            d_z = w.dot(hit_z[tower_labels == label])
            norm = np.sqrt(d_x ** 2 + d_y ** 2 + d_z ** 2)

            builder.field("E")
            builder.real(energy)
            builder.field("px")
            builder.real(energy * d_x / norm)
            builder.field("py")
            builder.real(energy * d_y / norm)
            builder.field("pz")
            builder.real(energy * d_z / norm)

            builder.end_record()
        builder.end_list()

        if debug:
           for w in weights:
               xs = x[tower_labels == label][w > 0.5]
               ys = y[tower_labels == label][w > 0.5]
```

```
                    padding = 0.02
                    for _x, _y in zip(xs, ys):
                        vis_kwargs = dict(
                            fill=False,
                            ls=["-", "--", ":"][cluster_ix % 3],
                            edgecolor=f"C{subcluster_ix}",
                            lw=1,
                            alpha=0.7,
                        )
                        ax[0].add_patch(mpl.patches.Rectangle(
                            (_x + padding, _y + padding), 1 - 2 * padding, 1 - 2 *
                            ↪  padding,
                            **vis_kwargs
                        ))
                        ax[1].add_patch(mpl.patches.Rectangle(
                            (wraparound_x(_x) + padding, _y + padding), 1 - 2 * padding,
                            ↪  1 - 2 * padding,
                            **vis_kwargs
                        ))
                    subcluster_ix += 1
                builder.end_record()
            builder.end_list()

    # Define schema for empty output case using a
    # workaround for https://github.com/scikit-hep/uproot5/issues/822
    builder.begin_list()
    with builder.record("Momentum4D"):
        for field in ["E", "px", "py", "pz", "lambda"]:
            builder.field(field)
            builder.real(np.nan)
        builder.field("subclusters")
        builder.begin_list()
        with builder.record("Momentum4D"):
            for field in ["E", "px", "py", "pz"]:
                builder.field(field)
                builder.real(np.nan)
        builder.end_list()
    builder.end_list()

    result = builder.snapshot()[:-1]

    return (result, ax) if debug else result


@dask.delayed(pure=True)
def find_island_cluster(events, batch_size=1000, **kwargs):
    import numpy as np
    import awkward as ak
    num_batches = int(np.ceil(len(events) / float(batch_size)))
    f = dask.delayed(_find_island_cluster, pure=True)
    batches = [
        f(events[batch * batch_size:(batch + 1) * batch_size], **kwargs)
        for batch in range(num_batches)
```

```
    ]
    with worker_client() as client:
        batches = client.gather(client.compute(batches))
    return ak.concatenate(batches)


_events = ak.Array([
    {
        "EcalBarrelSciGlassHits.cellID" : [0 << 16, 1 << 16, 2 << 16],
        "EcalBarrelSciGlassHits.energy" : [321, 123, 321],
        "EcalBarrelSciGlassHits.position.x" : [0, -1, -2],
        "EcalBarrelSciGlassHits.position.y" : [0, -1, -2],
        "EcalBarrelSciGlassHits.position.z" : [0, -1, -2],
    },
    {
        "EcalBarrelSciGlassHits.cellID" : [1 << 16, 3 << 16],
        "EcalBarrelSciGlassHits.energy" : [123, 321],
        "EcalBarrelSciGlassHits.position.x" : [-1, -2],
        "EcalBarrelSciGlassHits.position.y" : [-1, -2],
        "EcalBarrelSciGlassHits.position.z" : [-1, -2],
    },
])

np.testing.assert_equal(ak.num(_find_island_cluster(_events[:0]), axis=0), 0.)
# check that the schema workaround works
np.testing.assert_equal(str(_find_island_cluster(_events[:0]).type),
↪  str(_find_island_cluster(_events)[:0].type))

_clusters = _find_island_cluster(_events)
np.testing.assert_equal(
    _clusters["E"][0].to_numpy(),
    [321 + 123 + 321]
)
plt.show()
np.testing.assert_equal(
    _clusters["E"][1].to_numpy(),
    [123, 321]
)
# check number of clusters found
np.testing.assert_equal(
    ak.num(_clusters["E"], axis=1).to_numpy(),
    [1, 2]
)
# check number of subclusters found
np.testing.assert_equal(
    ak.num(_clusters["subclusters", "E"], axis=2)[0].to_numpy(),
    [2]
)
np.testing.assert_equal(
    ak.num(_clusters["subclusters", "E"], axis=2)[1].to_numpy(),
    [1, 1]
)
```

This is a distribution of λ parameter for photon clusters:

```
particle = "gamma"
p_thrown = 2. * GeV

simu = client.compute(run_ddsim(particle=particle, p_min=p_thrown, p_max=p_thrown,
↪  **common_sim_params))
clusters = client.compute(find_island_cluster(simu)).result()

output_dir = Path("results/find_island_cluster")
output_dir.mkdir(parents=True, exist_ok=True)

plt.hist(ak.flatten(clusters["lambda"]).to_numpy(), bins=100)
plt.title(rf"{particle} at $|p_{{\mathrm{{thrown}}}}| = {p_thrown}$ GeV")
plt.xlabel("$\lambda$", loc="right")
plt.savefig(output_dir / "lambda.pdf")
plt.show()
```



6.3 Energy resolution

energy_resolution

```
output_dir.mkdir(parents=True, exist_ok=True)

sigma_FWHM_cb = [0] * len(sim_settings)
scale_cb = [0] * len(sim_settings)
loc_cb = [0] * len(sim_settings)
fraction_below = [0] * len(sim_settings)

for ix, setting in enumerate(sim_settings):
    label = setting["label"]
    visual_params = setting["visual"]
    counts = histograms[p_thrown][ix][particle].result()

    import scipy.stats
    def f(x, n, beta, m, loc, scale):
        return n * scipy.stats.crystalball.pdf(x, beta, m, loc, scale)
    p0 = (np.sum(counts[10:]), 2., 3., 0.95, 0.05)
    try:
        import scipy.optimize
        par, pcov = scipy.optimize.curve_fit(f, e_over_p[5:], counts[5:], p0=p0,
        ↪  maxfev=10000)
    except RuntimeError:
        par = None

    if par is not None:
```

```
        _, _, _, loc_cb[ix], scale_cb[ix] = par

        # Calculate FWHM
        y_max = np.max(f(np.linspace(0., 1., 100), *par))
        f_prime = lambda x: f(x, *par) - y_max / 2
        x_plus, = scipy.optimize.root(f_prime, loc_cb[ix] + scale_cb[ix]).x
        x_minus, = scipy.optimize.root(f_prime, loc_cb[ix] - scale_cb[ix]).x
        fwhm = x_plus - x_minus
        sigma_FWHM_cb[ix] = fwhm / 2 / np.sqrt(2 * np.log(2))

        cutoff_x = loc_cb[ix] - fwhm
        fraction_below[ix] = np.sum(counts[e_over_p < cutoff_x]) /
        ↪  setting["sim"]["num_events"]

    fig = plt.figure(figsize=np.array(mpl.rcParams["figure.figsize"]) * 0.65)
    plt.stairs(
        counts, bins_e_over_p,
        label=f"$e^-$ ({label})", lw=2, ls=":", **visual_params,
    )

    if par is not None:
        plt.plot(e_over_p, f(e_over_p, *par), label=rf"Crystal Ball",
        ↪  color="tab:green", lw=0.8)


        if cutoff_x > 0:
            plt.axvline(x=loc_cb[ix], ls="--", color="tab:blue", lw=0.8)
            plt.text(
                loc_cb[ix], np.max(counts) / 2, "$\mu$", color="tab:blue",
                horizontalalignment="left", verticalalignment="center",
            )
            plt.axvline(x=cutoff_x, ls="--", color="tab:orange", lw=0.8)
            plt.text(
                cutoff_x, np.max(counts) / 2, "$\mu - \mathrm{FWHM}$",
                ↪  color="tab:orange",
                horizontalalignment="right", verticalalignment="center",
            )

    plt.legend()
    plt.title(rf"{particle} at $|p_{{\mathrm{{thrown}}}}| = {p_thrown}$ GeV")
    plt.xlabel(r"$E_{\mathrm{clust.}} / |p_{{\mathrm{{thrown}}}}|$", loc="right")
    plt.ylabel("Number of events", loc="top")
    plt.minorticks_on()
    plt.ylim(bottom=0.5)
    plt.savefig(output_dir / f"e_over_p_{particle}_{p_thrown:.2f}_{ix}.pdf",
    ↪  bbox_inches="tight")
    plt.show()
```

resolution_vs_pt

```
sim = {}
for p_thrown in energies:
    <<book_sim>>
```

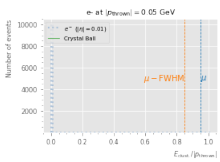
```
bins_e_over_p = np.linspace(0., 1.01, 102)
e_over_p = (bins_e_over_p[1:] + bins_e_over_p[:-1]) / 2

@dask.delayed(pure=True)
def to_histogram(events, clusters):
    p_thrown = np.sqrt(
        events["MCParticles.momentum.x"][:,0] ** 2
        + events["MCParticles.momentum.y"][:,0] ** 2
        + events["MCParticles.momentum.z"][:,0] ** 2
    )
    energies = ak.fill_none(ak.firsts(ak.sort(clusters.E, axis=-1, ascending=False),
    ↪  axis=-1), 0)
    vals = energies / p_thrown
    counts, _ = np.histogram(vals, bins=bins_e_over_p)
    return counts

histograms = {}
for p_thrown in energies:
    for ix, setting in enumerate(sim_settings):
        for particle in particles:
            events = sim[p_thrown][ix][particle]
            events = setting.get("event_filter", lambda events: events)(events)
            clusters = client.compute(find_island_cluster(events))
            future = client.compute(to_histogram(events, clusters))
            histograms.setdefault(p_thrown, {}).setdefault(ix, {})[particle] = future

E_res_cb_FWHM = []
E_res_cb_scale = []
_fraction_below = []
for p_thrown in energies:
    <<energy_resolution>>

    E_res_cb_FWHM.append(np.array(sigma_FWHM_cb) / np.array(loc_cb))
    E_res_cb_scale.append(np.array(scale_cb) / np.array(loc_cb))
    _fraction_below.append(fraction_below)

E_res_cb_FWHM = np.array(E_res_cb_FWHM).T
E_res_cb_scale = np.array(E_res_cb_scale).T
_fraction_below = np.array(_fraction_below).T


output_dir.mkdir(parents=True, exist_ok=True)

for ix, setting in enumerate(sim_settings):
    label = setting["label"]
    visual_params = setting["visual"]
    line, = plt.plot(energies, E_res_cb_FWHM[ix] * 100, label=label, **visual_params)

plt.fill_between(
    energies,
    np.sqrt((10 / np.sqrt(np.array(energies))) ** 2 + 1 ** 2),
    np.sqrt((12 / np.sqrt(np.array(energies))) ** 2 + 3 ** 2),
    alpha=0.5, color="black", label=r"YR requirement $10-12\% / \sqrt{E} \oplus
    ↪  1-3\%$",
```

```
)

plt.title(r"FWHM${}_{\mathrm{Crystal\ Ball}} / (2 \sqrt{2 \mathrm{log}(2)}
↪  \mu_{\mathrm{Crystal\ Ball}})$")
plt.xlabel(r"$|p_{\mathrm{thrown}}|$, GeV", loc="right")
plt.ylabel(r"$\sigma \{E\} / E$, %", loc="top")
plt.legend()
plt.ylim(bottom=0., top=15.)
plt.savefig(output_dir / "resolution_FWHM.pdf")
plt.show()

for ix, setting in enumerate(sim_settings):
    label = setting["label"]
    visual_params = setting["visual"]
    line, = plt.plot(energies, E_res_cb_scale[ix] * 100, label=label, **visual_params)

plt.title(r"$\sigma_{\mathrm{Crystal\ Ball}} / \mu_{\mathrm{Crystal\ Ball}}$")
plt.xlabel(r"$|p_{\mathrm{thrown}}|$, GeV", loc="right")
plt.ylabel(r"$\sigma \{E\} / E$, %", loc="top")
plt.legend()
plt.ylim(bottom=0., top=3.1)
plt.savefig(output_dir / "resolution_sigma.pdf")
plt.show()

for ix, setting in enumerate(sim_settings):
    label = setting["label"]
    visual_params = setting["visual"]
    line, = plt.plot(energies, _fraction_below[ix] * 100, label=label, **visual_params)

plt.title(r"Fraction of particles with $E/p < \mu - \mathrm{FWHM}$")
plt.xlabel(r"$|p_{\mathrm{thrown}}|$, GeV", loc="right")
plt.ylabel(r"Fraction, %", loc="top")
plt.legend()
plt.savefig(output_dir / "percent_low.pdf")
plt.show()
```

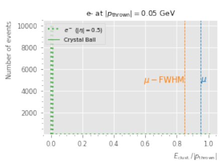
```
<<sim_settings>>
particles = ["e-"]
particle = "e-"

apply_energy_threshold_delayed = dask.delayed(apply_energy_threshold, pure=True)
sim_settings = [
    dict(
        event_filter=lambda events: apply_energy_threshold_delayed(events, 50 * MeV),
        sim=dict(
            common_sim_params,
            abseta_min=eta,
            abseta_max=eta,
            theta_min=None,
            theta_max=None,
        ),
        label=rf"$|\eta| = {eta}$",
```

```
        visual=dict(
            color=color,
        ),
    ) for eta, color in zip([0.01, 0.5, 1.], list(mpl.colors.XKCD_COLORS))
]

output_dir = Path("results/charge_eta_bins")

#for s in sim_settings:
#    s["sim"]["num_events"] = NUM_EVENTS_DEFAULT * 10

<<resolution_vs_pt>>
```

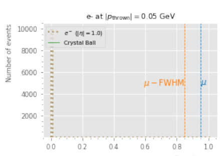
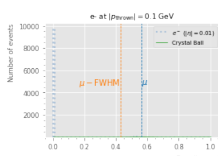
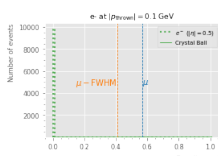
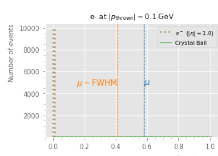
```
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/s
  warnings.warn('Covariance of the parameters could not be estimated',
```



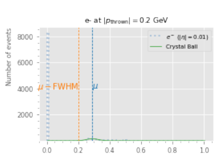
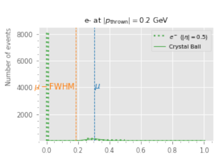
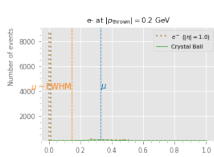
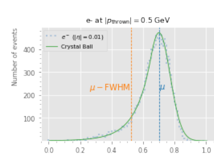
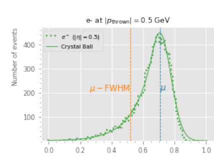
```
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/s
  warnings.warn('Covariance of the parameters could not be estimated',
```



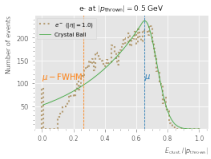
```
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/s
  warnings.warn('Covariance of the parameters could not be estimated',
```

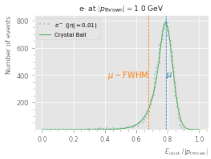
```
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/s
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/s
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
```

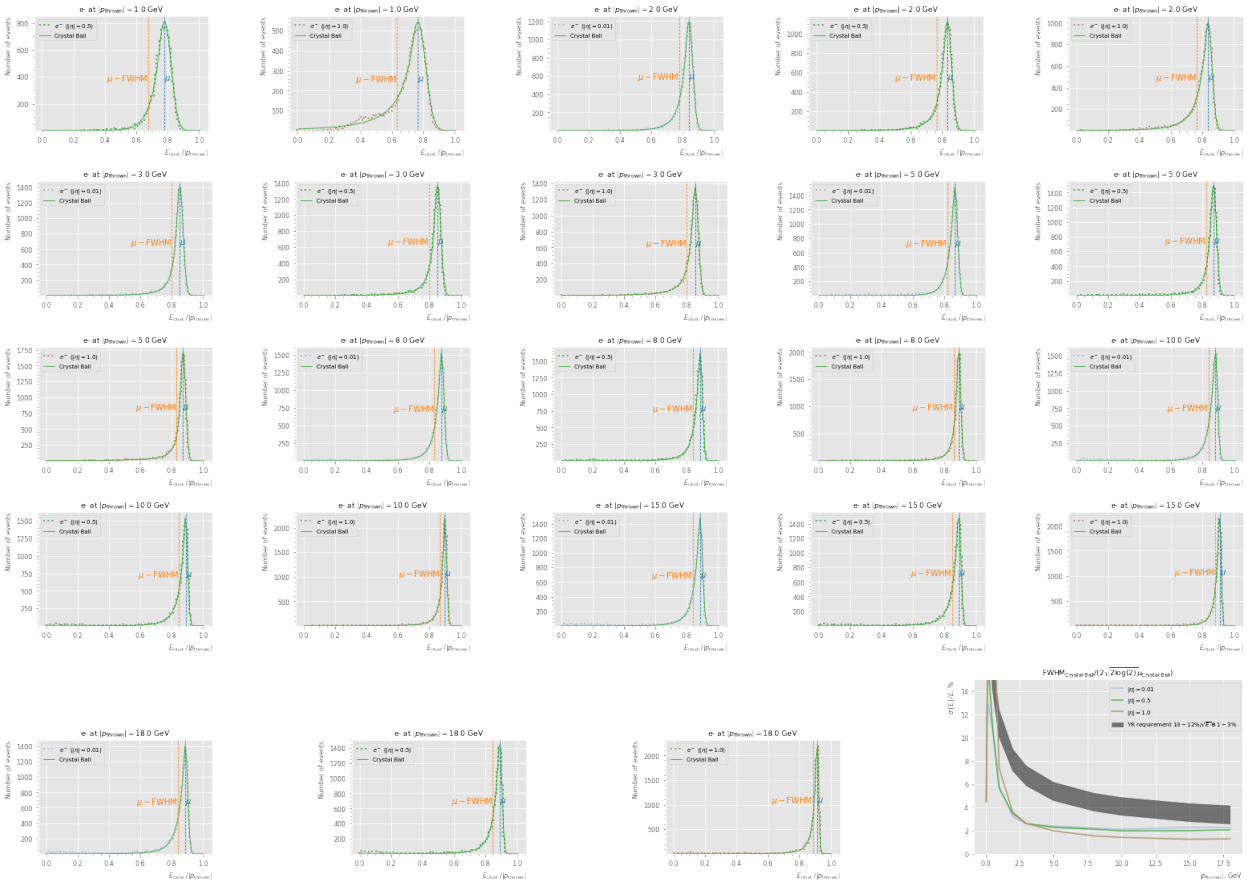
```
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/s
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/s
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
```

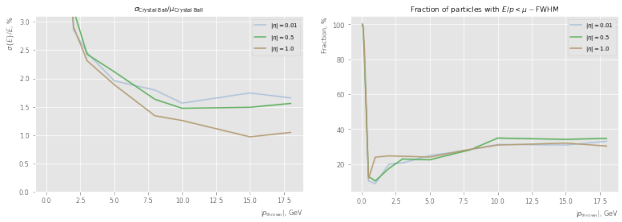


```
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/s
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/s
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
```



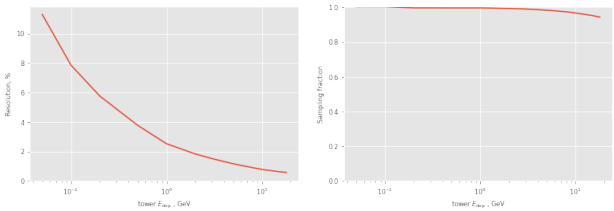
```
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/s
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/s
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
```





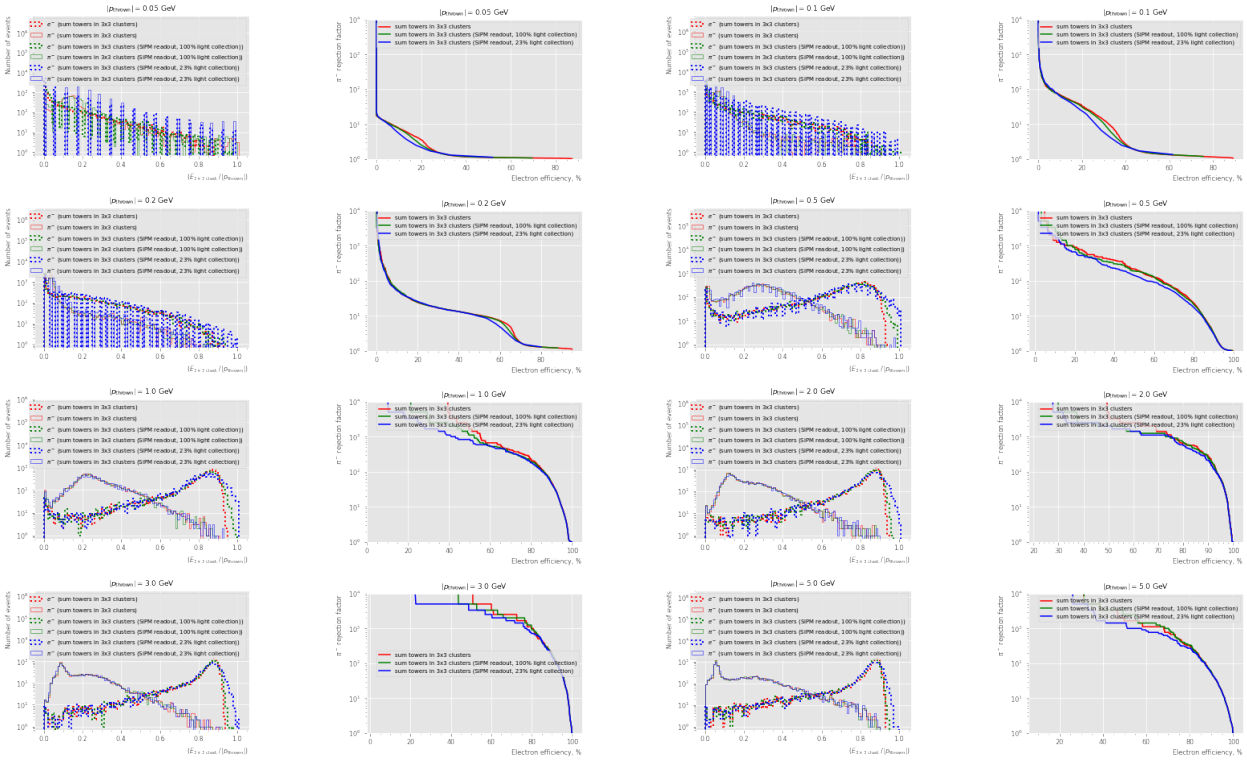
6.3.1 SiPM occupancy and light collection

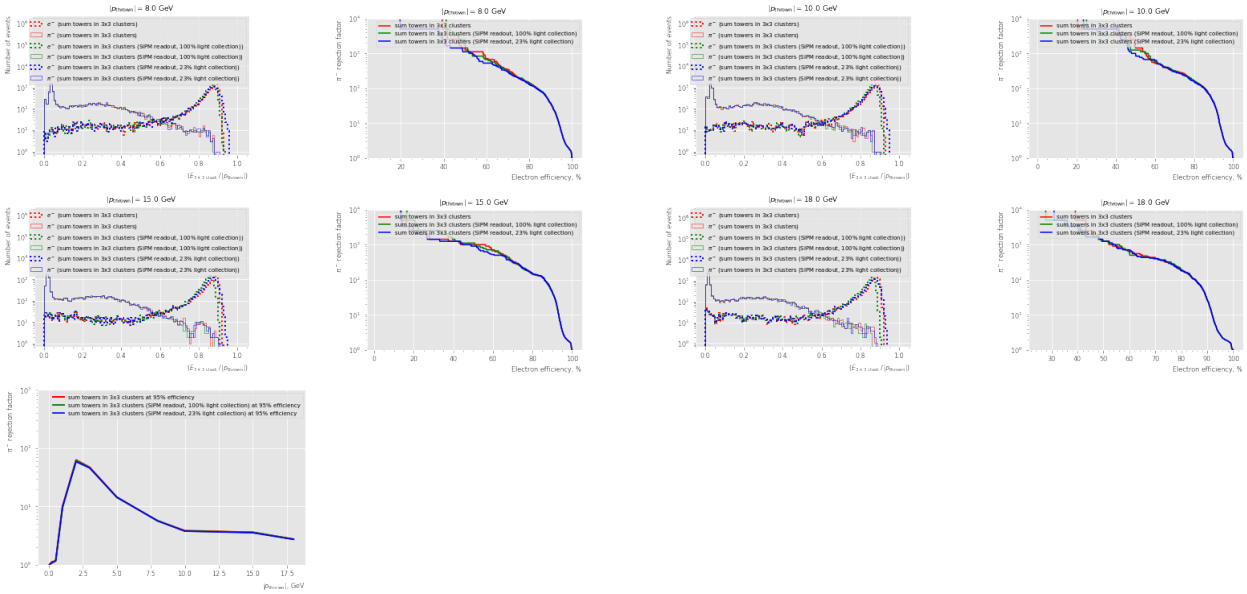
```
energies = [0.05, 0.1, 0.2, 0.5, 1., 2., 3., 5., 8., 10., 15., 18.]

QUANTUM_EFF = 0.5
PHOTONS_PER_GEV = 3000
NUM_MPPCs = 16
LIGHT_COLLECTION_EFF = 1 # 6 * 6 * NUM_MPPCs / (50 * 60)
PIXELS_PER_MPPC = 14331
N_pixels = NUM_MPPCs * PIXELS_PER_MPPC

sampling_fractions = []
resolutions = []

rng = np.random.default_rng(seed=1)

for e_dep in energies:
    P_pixel = 1. - np.exp(-np.array(e_dep) * PHOTONS_PER_GEV * QUANTUM_EFF *
    ↪  LIGHT_COLLECTION_EFF / N_pixels)

    e_meas = rng.binomial(N_pixels, P_pixel, size=1000) / (PHOTONS_PER_GEV *
    ↪  QUANTUM_EFF * LIGHT_COLLECTION_EFF)

    sampling_fractions.append(np.mean(e_meas) / e_dep)
    resolutions.append(np.std(e_meas) / np.mean(e_meas))

plt.plot(energies, np.array(resolutions) * 100)
plt.ylim(bottom=0.)
plt.xscale("log")
plt.xlabel("tower $E_{\mathrm{dep.}}$, GeV")
plt.ylabel("Resolution, %")
plt.show()

plt.plot(energies, np.array(sampling_fractions))
plt.ylim(bottom=0.)
plt.xscale("log")
plt.xlabel("tower $E_{\mathrm{dep.}}$, GeV")
plt.ylabel("Sampling fraction")
plt.show()
```



```
<<sim_settings>>


def apply_sipm_readout(events, light_collection_efficiency):
    rng = np.random.default_rng(seed=1)

    e_dep = ak.flatten(events["EcalBarrelSciGlassHits.energy"])

    P_pixel = 1. - np.exp(-np.array(e_dep) * PHOTONS_PER_GEV * QUANTUM_EFF *
    ↪  light_collection_efficiency / N_pixels)

    e_meas = rng.binomial(N_pixels, P_pixel) / (PHOTONS_PER_GEV * QUANTUM_EFF *
    ↪  light_collection_efficiency)

    counts = ak.num(events["EcalBarrelSciGlassHits.energy"])
    e_meas = ak.unflatten(e_meas, counts)

    return ak.Array({
        field: e_meas if field == "EcalBarrelSciGlassHits.energy" else events[field]
        for field in events.fields
    })


def enable_sipm_readout(classifier, light_collection_efficiency=1.):
    def new_classifier(events):
        return classifier(apply_sipm_readout(events, light_collection_efficiency))
    if hasattr(classifier, "xlabel"):
        new_classifier.xlabel = classifier.xlabel.replace(r"E_{\mathrm{dep.}}",
        ↪  r"E_{\mathrm{meas.}}")
    return new_classifier


LIGHT_COLLECTION_EFFICIENCY = 6 * 6 * NUM_MPPCs / (50 * 50)


sim_settings = [
    dict(
        classifier=edep_cluster_sum_classifier(size=3),
        sim=dict(
            common_sim_params,
        ),
        label="sum towers in 3x3 clusters",
        visual=dict(
            color="red",
        ),
    ),
```

```
    dict(
        classifier=enable_sipm_readout(edep_cluster_sum_classifier(size=3),
        ↪  light_collection_efficiency=1.),
        sim=dict(
            common_sim_params,
        ),
        label="sum towers in 3x3 clusters (SiPM readout, 100% light collection)",
        visual=dict(
            color="green",
        ),
    ),
    dict(
        classifier=enable_sipm_readout(edep_cluster_sum_classifier(size=3),
        ↪  light_collection_efficiency=LIGHT_COLLECTION_EFFICIENCY),
        sim=dict(
            common_sim_params,
        ),
        label=f"sum towers in 3x3 clusters (SiPM readout, {LIGHT_COLLECTION_EFFICIENCY
        ↪  * 100:.0f}% light collection)",
        visual=dict(
            color="blue",
        ),
    ),
]

output_dir = Path("results/sipm_readout")

<<pion_rej_vs_pt>>
```

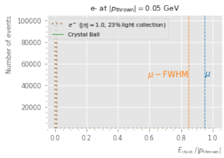
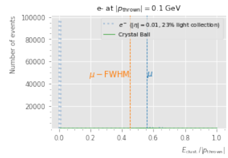
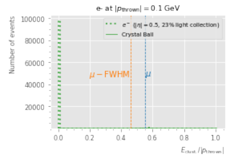
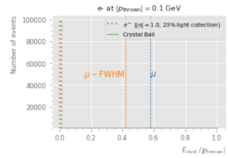




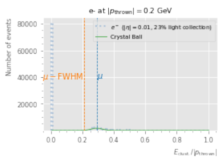
```
<<sim_settings>>
particles = ["e-"]
particle = "e-"

LIGHT_COLLECTION_EFFICIENCY = 6 * 6 * NUM_MPPCs / (50 * 50)

apply_energy_threshold_delayed = dask.delayed(apply_energy_threshold, pure=True)
apply_sipm_readout_delayed = dask.delayed(apply_sipm_readout, pure=True)
sim_settings = [
    dict(
        event_filter=lambda events:
        ↪  apply_energy_threshold_delayed(apply_sipm_readout_delayed(events,
        ↪  light_collection_efficiency=LIGHT_COLLECTION_EFFICIENCY), 50 * MeV),
        sim=dict(
            common_sim_params,
            abseta_min=eta,
            abseta_max=eta,
            theta_min=None,
            theta_max=None,
        ),
        label=rf"$|\eta| = {eta}$, {LIGHT_COLLECTION_EFFICIENCY * 100:.0f}% light
        ↪  collection",
        visual=dict(
            color=color,
        ),
    ) for eta, color in zip([0.01, 0.5, 1.], list(mpl.colors.XKCD_COLORS))
]

output_dir = Path("results/sipm_readout")

for s in sim_settings:
    s["sim"]["num_events"] = NUM_EVENTS_DEFAULT * 10
```

```
<<resolution_vs_pt>>
```

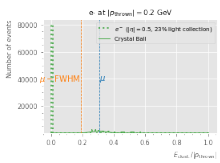
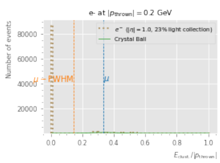
```
87a8595e-51c3-4652-96ae-fc08b1fc23c3/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-py
  warnings.warn('Covariance of the parameters could not be estimated',
```

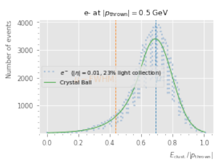
```
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
```



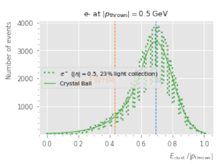
```
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
```

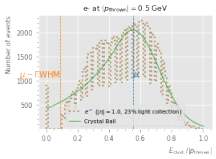
```
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
```



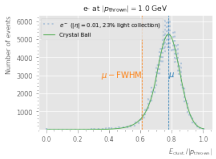
```
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
```



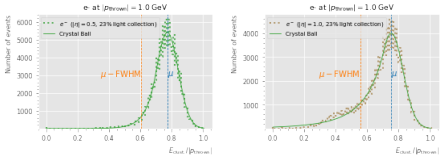
```
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/s
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/s
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
```



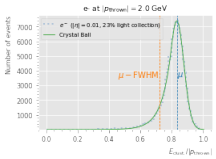

```
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/s
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/s
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
```



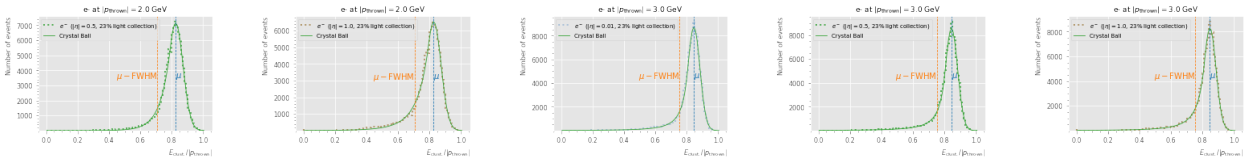

```
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/s
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/s
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
```

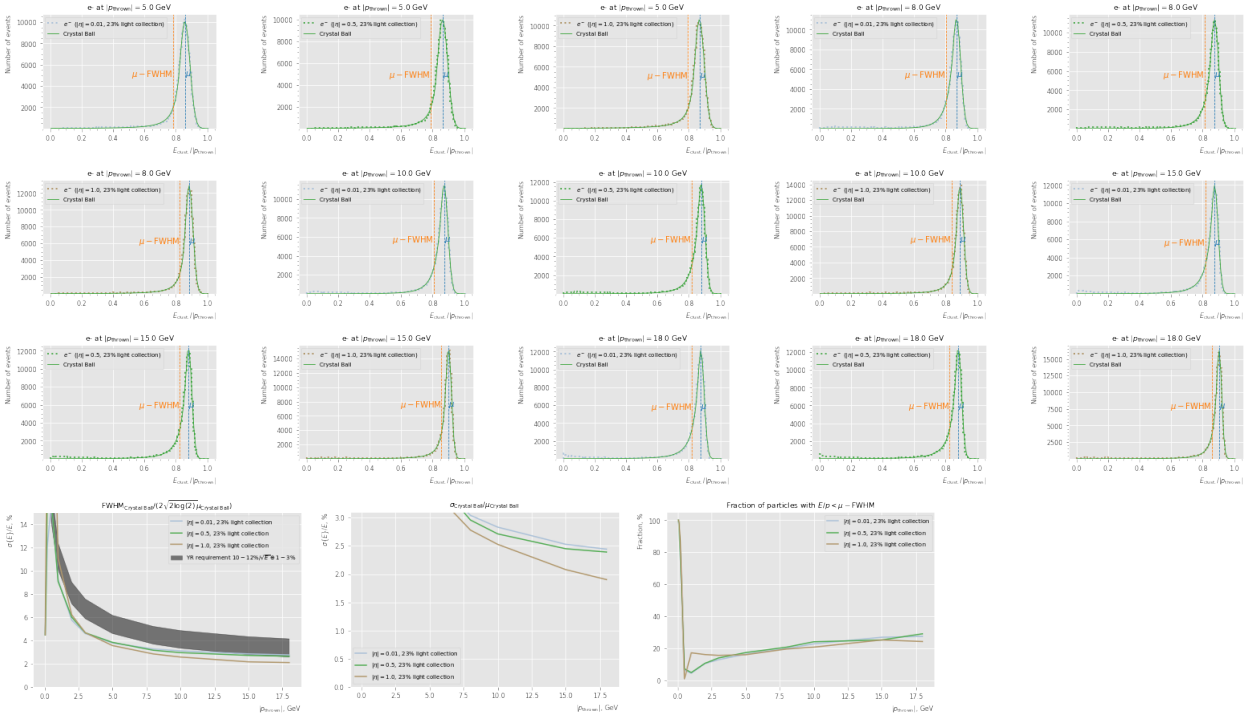



```
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/s
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/s
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
```



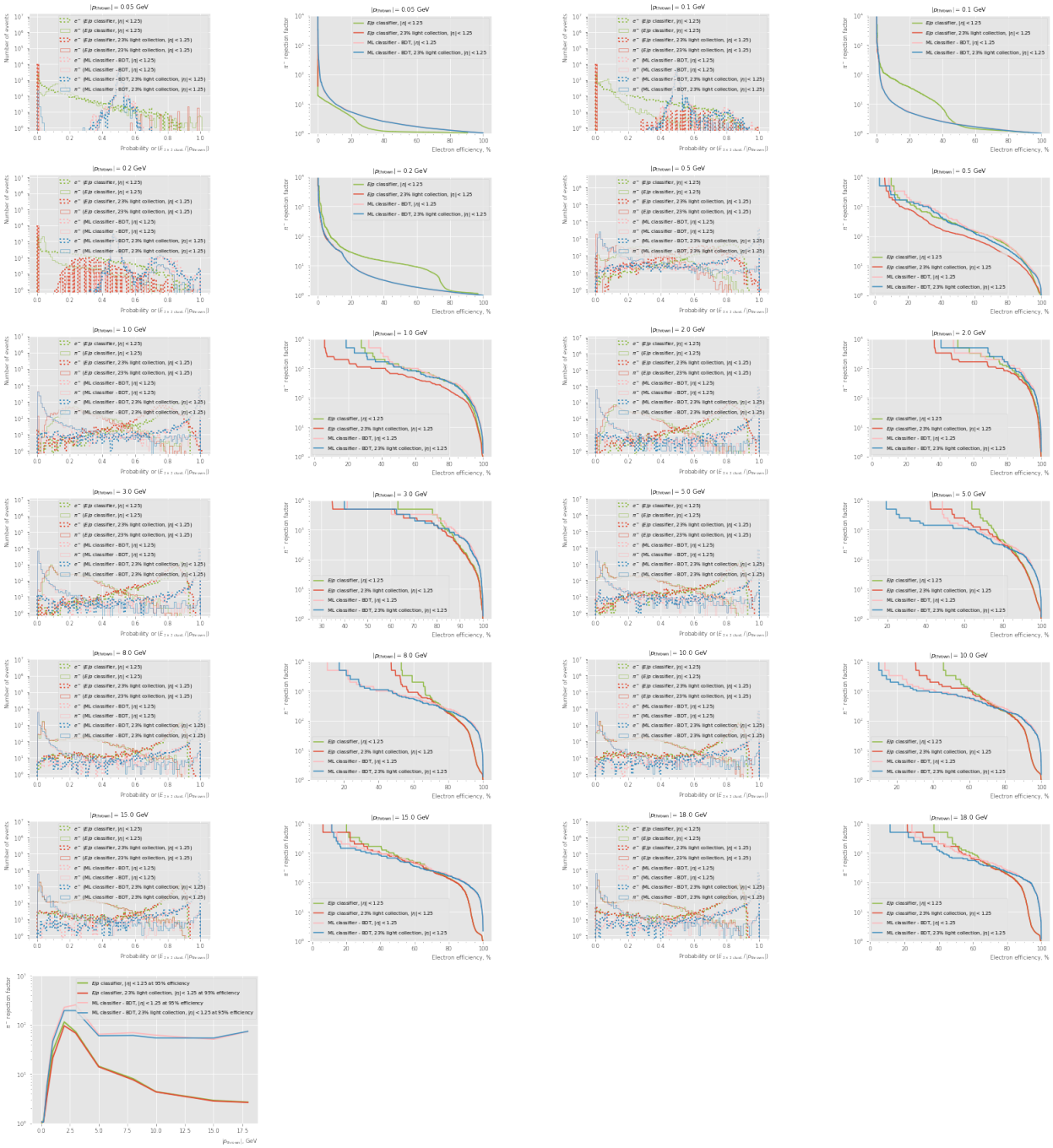

```
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/s
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
/project/rhfate2_uksr/nix/store/m79yjyvb2wrnqfk6651gsfi0mw0x854l-python3.10-scipy-1.8.1/lib/python3.10/s
  return ((m/beta)**m * np.exp(-beta**2 / 2.0) *
```





```
LIGHT_COLLECTION_EFFICIENCY = 6 * 6 * NUM_MPPCs / (50 * 50)

<<ml_sim_settings>>

particles = ["e-", "pi-"]

sim = {}
for p_thrown in energies:
    <<book_sim>>

sim_train = sim
sim = {}

from xgboost import XGBClassifier

<<train_def>>
<<to_x_with_3momentum_def>>

xgboost_with_p_vec_classifier_sipm = train(
    [sim_train[p_thrown][0]["e-"] for p_thrown in energies],
    [sim_train[p_thrown][0]["pi-"] for p_thrown in energies],
    enable_energy_threshold(enable_sipm_readout(to_x_with_3momentum,
    ↪  light_collection_efficiency=LIGHT_COLLECTION_EFFICIENCY), 50 * MeV),
    XGBClassifier(),
)

<<sim_settings>>

apply_energy_threshold_delayed = dask.delayed(apply_energy_threshold, pure=True)
apply_sipm_readout_delayed = dask.delayed(apply_sipm_readout, pure=True)
```

```
sim_settings = [
    dict(
        classifier=classifier, # no need to enable sipm/threshold twice
        event_filter=lambda events:
        ↪  apply_energy_threshold_delayed(apply_sipm_readout_delayed(events,
        ↪  light_collection_efficiency=LIGHT_COLLECTION_EFFICIENCY), 50 * MeV),
        sim=dict(
            common_sim_params,
            abseta_min=eta_min,
            abseta_max=eta_max,
            theta_min=None,
            theta_max=None,
        ),
        label=rf"{label}, $|\eta| < {eta_max}$",
        visual=dict(
            color=color,
        ),
    ) for eta_min, eta_max, classifier, label, color in [
        (0., 1.25, edep_cluster_sum_classifier(size=3), "$E/p$ classifier", "C5"),
        (0., 1.25,
        ↪  enable_energy_threshold(enable_sipm_readout(edep_cluster_sum_classifier(size=3),
        ↪  light_collection_efficiency=LIGHT_COLLECTION_EFFICIENCY), 50 * MeV),
        ↪  f"$E/p$ classifier, {LIGHT_COLLECTION_EFFICIENCY * 100:.0f}% light
        ↪  collection", "C0"),
        (0., 1.25, xgboost_with_p_vec_classifier, "ML classifier - BDT", "C6"),
        (0., 1.25, xgboost_with_p_vec_classifier_sipm, f"ML classifier - BDT,
        ↪  {LIGHT_COLLECTION_EFFICIENCY * 100:.0f}% light collection", "C1"),
    ]
]

output_dir = Path("results/sipm_readout")

#particles = ["e-"]
#particle = "e-"
#
#for s in sim_settings:
#    s["sim"]["num_events"] = NUM_EVENTS_DEFAULT * 10
#
#<<resolution_vs_pt>>
<<pion_rej_vs_pt>>

#print("\n".join(["\t".join(map(str, row)) for row in np.stack([energies,
↪  E_res_cb_FWHM[0]]).T]))

for row in np.array(list(zip(energies, *[1 / np.array([fpr(0.95) for fpr in roc[ix]])
↪  for ix, setting in enumerate(sim_settings)]))):
    print("\t".join(map(lambda val: f"{val:.1f}", row)))
```



0.1 1.0 1.1 1.1 1.1
0.1 1.0 1.1 1.1 1.0
0.2 1.2 1.1 1.1 1.1
0.5 4.2 3.5 6.8 5.9
1.0 29.5 21.2 53.8 45.9
2.0 114.9 96.2 227.3 196.1
3.0 72.5 68.0 256.4 196.1
5.0 14.5 14.1 64.5 60.2
8.0 8.1 7.6 69.4 61.0
10.0 4.4 4.3 61.7 54.3
15.0 2.9 2.8 50.8 54.1

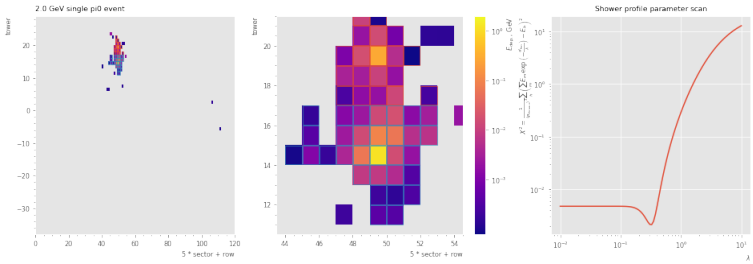
```
18.0 2.7 2.7 73.5 74.1
```

6.4 Neutral pion reconstruction

6.4.1 Example clusters

```
particle = "pi0"
p_thrown = 2. * GeV

output_dir = Path("results/pi0_reconstruction")
output_dir.mkdir(parents=True, exist_ok=True)

simu = client.compute(run_ddsim(particle=particle, p_min=p_thrown, p_max=p_thrown,
↪  **dict(common_sim_params, num_events=2)))
%time simu.result()

_, ax = _find_island_cluster(simu.result()[1:2], debug=True)
plt.sca(ax[0])
plt.title(f"{p_thrown} GeV single {particle} event", loc="left")
plt.savefig(output_dir / f"event_{particle}_{p_thrown:.2f}.pdf")
plt.show()
```

```
CPU times: user 39 s, sys: 2.49 s, total: 41.5 s
Wall time: 1min
[0.28297442 1.1490308 ]
```

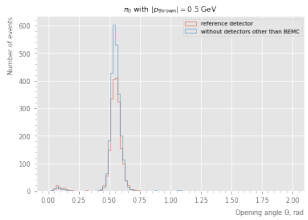



```
<<sim_settings>>
particles = ["pi0"]
particle = "pi0"
#p_thrown = 2. * GeV
#energies = [2. * GeV]

from copy import deepcopy

energies = [p_thrown for p_thrown in energies if p_thrown >= 0.5]
sim_settings = [
    dict(
        sim=dict(
            common_sim_params,
        ),
        label="reference detector",
        visual=dict(
            color="C0",
```

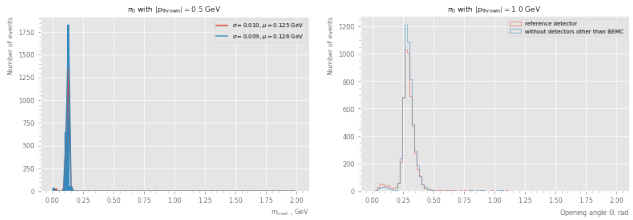
```
        ),
    ),
    dict(
        sim=dict(
            common_sim_params,
            detector_config="epic_sciglass_only",
        ),
        label="without detectors other than BEMC",
        visual=dict(
            color="C1",
        ),
    ),
]

sim = {}
clusters = {}
for p_thrown in energies:
    <<book_sim>>
    for ix, setting in enumerate(sim_settings):
        for particle in particles:
            clusters.setdefault(p_thrown, {}).setdefault(ix, {})[particle] =
            ↪  client.compute(find_island_cluster(sim[p_thrown][ix][particle]))

output_dir = Path("results/pi0_reconstruction")
output_dir.mkdir(parents=True, exist_ok=True)

for p_thrown in energies:
    for ix, setting in enumerate(sim_settings):
        energies_ix =
        ↪  ak.argsort(ak.flatten(clusters[p_thrown][ix]["pi0"].result()["subclusters",
        ↪  "E"], axis=-1), ascending=False, axis=-1)
        p1 =
        ↪  ak.firsts(ak.flatten(clusters[p_thrown][ix]["pi0"].result()["subclusters"],
        ↪  axis=-1)[energies_ix], axis=-1)
        p2 =
        ↪  ak.firsts(ak.flatten(clusters[p_thrown][ix]["pi0"].result()["subclusters"],
        ↪  axis=-1)[energies_ix[...,1:]], axis=-1)

        p1 = p1[~ak.is_none(p2)]
        p2 = p2[~ak.is_none(p2)]

        label = setting["label"]
        visual_params = setting["visual"]

        theta = np.arccos(p1.to_Vector3D().dot(p2.to_Vector3D()) / p1.to_Vector3D().mag
        ↪  / p2.to_Vector3D().mag).to_numpy()
        plt.hist(theta, bins=np.linspace(0., 2., 100), histtype="step", label=label,
        ↪  **visual_params)
        np.testing.assert_array_less(np.zeros_like(theta), theta)
        np.testing.assert_array_less(theta, np.full_like(theta, 2.))

    plt.title(f"$\pi_0$ with $|p_{{\mathrm{{thrown}}}}| = {p_thrown}$ GeV")
    plt.xlabel("Opening angle $\Theta$, rad", loc="right")
```

```
    plt.ylabel(r"Number of events", loc="top")
    plt.minorticks_on()
    plt.legend()
    plt.savefig(output_dir / f"pi0_angle_{p_thrown:.2f}.pdf")
    plt.show()

    for ix, setting in enumerate(sim_settings):
        energies_ix =
        ↪  ak.argsort(ak.flatten(clusters[p_thrown][ix]["pi0"].result()["subclusters",
        ↪  "E"], axis=-1), ascending=False, axis=-1)
        p1 =
        ↪  ak.firsts(ak.flatten(clusters[p_thrown][ix]["pi0"].result()["subclusters"],
        ↪  axis=-1)[energies_ix], axis=-1)
        p2 =
        ↪  ak.firsts(ak.flatten(clusters[p_thrown][ix]["pi0"].result()["subclusters"],
        ↪  axis=-1)[energies_ix[...,1:]], axis=-1)

        p1 = p1[~ak.is_none(p2)]
        p2 = p2[~ak.is_none(p2)]

        label = setting["label"]
        visual_params = setting["visual"]

        def determine_calibration_constant(p_thrown):
            particle = "gamma"

            simu = client.compute(run_ddsim(particle=particle, p_min=p_thrown / 2.,
            ↪  p_max=p_thrown / 2., **common_sim_params))
            clusters = client.compute(find_island_cluster(simu)).result()

            counts, bins, _ = plt.hist(ak.firsts(ak.sort(clusters.E, axis=-1,
            ↪  ascending=False), axis=-1) / (p_thrown / 2.), bins=100)
            plt.show()

            e_over_p = (bins[1:] + bins[-1]) / 2
            return e_over_p[np.argmax(counts)]


        CALIBRATION_CONSTANT = 1 # determine_calibration_constant()
        print(f"Calibration constant {CALIBRATION_CONSTANT}")

        m = (p1 + p2).mass.to_numpy() / CALIBRATION_CONSTANT
        counts, bins, _ = plt.hist(m, bins=np.linspace(0., 2., 100))
        np.testing.assert_array_less(np.zeros_like(m), m)
        np.testing.assert_array_less(m, np.full_like(m, 10.))
        def f(x, mean, sigma, norm):
            return norm * np.exp(-((x - mean) / sigma) ** 2 / 2)
        import scipy.optimize
        xs = (bins[:-1] + bins[1:]) / 2
        par, pcov = scipy.optimize.curve_fit(f, xs, counts)
        mean, sigma, norm = par
        plt.plot(xs, f(xs, *par), label=rf"$\sigma = {sigma:.3f}$, $\mu = {mean:.3f}$
        ↪  GeV", **visual_params)
```

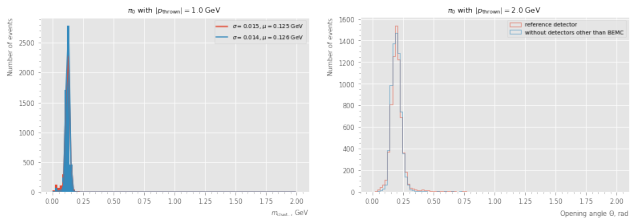
```
    plt.title(f"$\pi_0$ with $|p_{{\mathrm{{thrown}}}}| = {p_thrown}$ GeV")
    plt.xlabel(r"$m_{\mathrm{clust.}}$, GeV", loc="right")
    plt.ylabel(r"Number of events", loc="top")
    plt.minorticks_on()
    plt.legend()
    plt.savefig(output_dir / f"pi0_mass_{p_thrown:.2f}.pdf")
    plt.show()
```



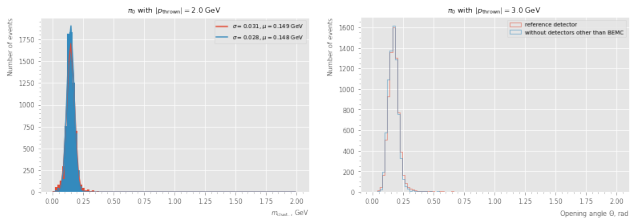
```
Calibration constant 1
Calibration constant 1
```



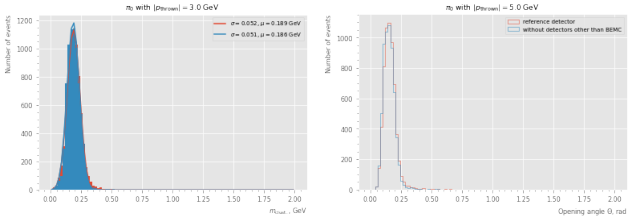
```
Calibration constant 1
Calibration constant 1
```



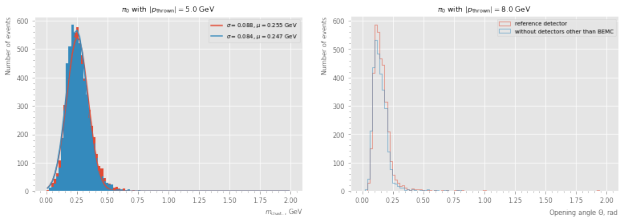
```
Calibration constant 1
Calibration constant 1
```



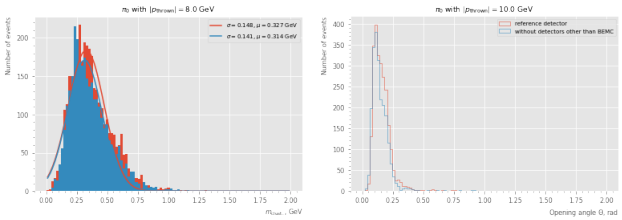
```
Calibration constant 1
Calibration constant 1
```



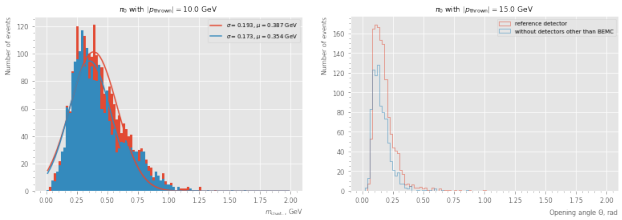
Calibration constant 1
Calibration constant 1



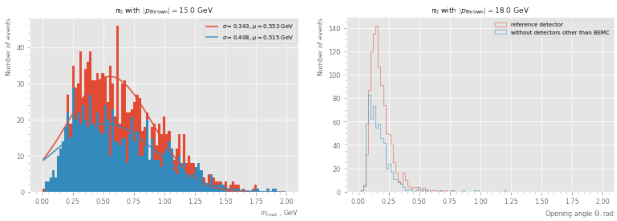
Calibration constant 1
Calibration constant 1



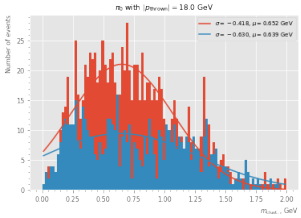
Calibration constant 1
Calibration constant 1



Calibration constant 1
Calibration constant 1



Calibration constant 1
Calibration constant 1



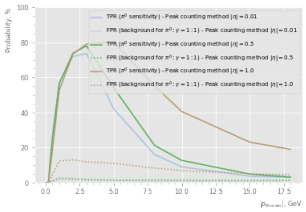
6.5 Separation of gamma from pi0 decay

```
<<sim_settings>>
particles = ["gamma", "pi0"]

sim_settings = [
    dict(
        sim=dict(
            common_sim_params,
            abseta_min=eta,
            abseta_max=eta,
            theta_min=None,
            theta_max=None,
        ),
        label=rf"$|\eta| = {eta}$",
        visual=dict(
            color=color,
        ),
    ) for eta, color in zip([0.01, 0.5, 1.], list(mpl.colors.XKCD_COLORS))
]

output_dir = Path("results/charge_eta_bins")
output_dir.mkdir(parents=True, exist_ok=True)

<<find_island_cluster_def>>

sim = {}
clusters = {}
for p_thrown in energies:
    <<book_sim>>
    for ix, setting in enumerate(sim_settings):
        for particle in particles:
            clusters.setdefault(p_thrown, {}).setdefault(ix, {})[particle] =
            ↪  client.compute(find_island_cluster(sim[p_thrown][ix][particle]))

for ix, setting in enumerate(sim_settings):
    label = setting["label"]
    visual_params = setting["visual"]

    tpr = []

    for p_thrown in energies:
        has_clusters = ak.num(clusters[p_thrown][ix]["pi0"].result(), axis=1) >= 1
        is_pi0 = ak.sum(ak.num(clusters[p_thrown][ix]["pi0"].result()["subclusters"],
        ↪  axis=2), axis=1) >= 2
```

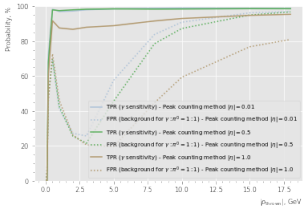
```
        tpr.append(ak.sum(is_pi0) / float(ak.sum(has_clusters)))

    plt.plot(energies, np.array(tpr) * 100, label=rf"TPR ($\pi^0$ sensitivity) - Peak
    ↪  counting method {label}", **visual_params)

    fpr = []

    for p_thrown in energies:
        has_clusters = ak.num(clusters[p_thrown][ix]["gamma"].result(), axis=1) >= 1
        try:
            is_pi0 =
            ↪  ak.sum(ak.num(clusters[p_thrown][ix]["gamma"].result()["subclusters"],
            ↪  axis=2), axis=1) >= 2
        except ValueError:
            is_pi0 = np.array([False])
        fpr.append(ak.sum(is_pi0) / float(ak.sum(has_clusters)))

    plt.plot(energies, np.array(fpr) * 100, label=rf"FPR (background for $\pi^0:\gamma
    ↪  = 1:1$) - Peak counting method {label}", ls=":", **visual_params)

plt.ylim(0., 100.)
plt.xlabel(r"$|p_{\mathrm{thrown}}|$, GeV", loc="right")
plt.ylabel("Probability, %", loc="top")
plt.minorticks_on()
plt.legend()
plt.savefig(output_dir / "pi0_separation.pdf", bbox_inches="tight")

plt.show()

for ix, setting in enumerate(sim_settings):
    label = setting["label"]
    visual_params = setting["visual"]

    tpr = []

    for p_thrown in energies:
        has_clusters = ak.num(clusters[p_thrown][ix]["gamma"].result(), axis=1) >= 1
        is_gamma =
        ↪  ak.sum(ak.num(clusters[p_thrown][ix]["gamma"].result()["subclusters"],
        ↪  axis=2), axis=1) == 1
        tpr.append(ak.sum(is_gamma) / float(ak.sum(has_clusters)))

    plt.plot(energies, np.array(tpr) * 100, label=rf"TPR ($\gamma$ sensitivity) - Peak
    ↪  counting method {label}", **visual_params)

    fpr = []

    for p_thrown in energies:
        has_clusters = ak.num(clusters[p_thrown][ix]["pi0"].result(), axis=1) >= 1
        try:
            is_gamma =
            ↪  ak.sum(ak.num(clusters[p_thrown][ix]["pi0"].result()["subclusters"],
            ↪  axis=2), axis=1) == 1
```

```
        except ValueError:
            is_gamma = np.array([False])
        fpr.append(ak.sum(is_gamma) / float(ak.sum(has_clusters)))

    plt.plot(energies, np.array(fpr) * 100, label=rf"FPR (background for $\gamma:\pi^0
    ↪  = 1:1$) - Peak counting method {label}", ls=":", **visual_params)

plt.ylim(0., 100.)
plt.xlabel(r"$|p_{\mathrm{thrown}}|$, GeV", loc="right")
plt.ylabel("Probability, %", loc="top")
plt.minorticks_on()
plt.legend()
plt.savefig(output_dir / "gamma_separation.pdf", bbox_inches="tight")

plt.show()
```

```
/tmp/nix-shell.cLxFME/ipykernel_65508/2047952802.py:564: RuntimeWarning: invalid value encountered in di
  fpr.append(ak.sum(is_pi0) / float(ak.sum(has_clusters)))
```



```
/tmp/nix-shell.cLxFME/ipykernel_65508/2047952802.py:586: RuntimeWarning: invalid value encountered in di
  tpr.append(ak.sum(is_gamma) / float(ak.sum(has_clusters)))
```



pion_gamma_discrimination

```
import numpy as np

bins_e_over_p = np.linspace(0., 1.01, 102)

fig = plt.figure(figsize=np.array(mpl.rcParams["figure.figsize"]) * 0.65)

y_pred = {}
for ix, setting in enumerate(sim_settings):
    classifier = setting["classifier"]
    label = setting["label"]
    visual_params = setting["visual"]
    y_pred.setdefault(ix, {})
    y_pred[ix]["pi0"] = classifier(sim[p_thrown][ix]["pi0"].result())
    y_pred[ix]["gamma"] = classifier(sim[p_thrown][ix]["gamma"].result())
    _, _, _ = plt.hist(y_pred[ix]["pi0"], bins=bins_e_over_p, label=f"$\pi^0$
    ↪  ({label})", histtype="step", lw=2, ls=":", **visual_params)
```

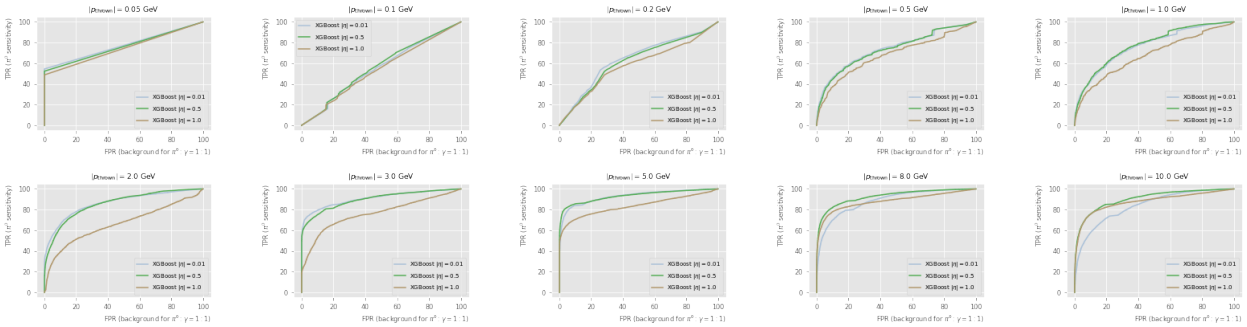
```
    _, _, _ = plt.hist(y_pred[ix]["gamma"], bins=bins_e_over_p, label=f"$\gamma$
    ↪  ({label})", histtype="step", **visual_params)

plt.clf()

fpr = {}
tpr = {}
for ix, setting in enumerate(sim_settings):
    label = setting["label"]
    visual_params = setting["visual"]

    from sklearn.metrics import roc_curve
    fpr[ix], tpr[ix], _ = roc_curve(
        np.concatenate([np.ones_like(y_pred[ix]["pi0"]),
        ↪  np.zeros_like(y_pred[ix]["gamma"])]),
        np.concatenate([y_pred[ix]["pi0"], y_pred[ix]["gamma"]]),
    )
    plt.plot(fpr[ix] * 100, tpr[ix] * 100, label=label, **visual_params)

plt.title(rf"$|p_{{\mathrm{{thrown}}}}| = {p_thrown}$ GeV")
plt.xlabel(r"FPR (background for $\pi^0:\gamma = 1:1$)", loc="right")
plt.ylabel(r"TPR ($\pi^0$ sensitivity)", loc="top")
plt.legend()
plt.savefig(output_dir / f"pion_gamma_discimination_{p_thrown:.2f}_ml.pdf",
↪  bbox_inches="tight")
plt.show()
```

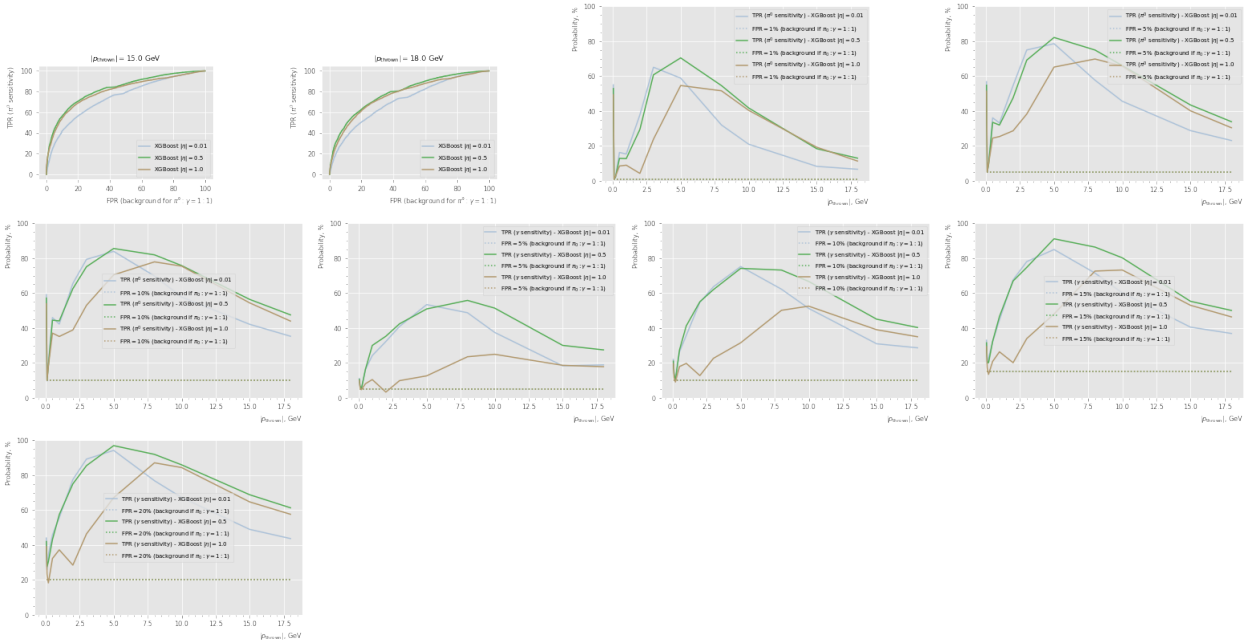
For Machine Learning, the setup is similar to the one for pion rejection, but we can't rely on the true momentum

```
<<sim_settings>>

sim_settings = [
    dict(
        sim=dict(
            common_sim_params,
            num_events=NUM_EVENTS_DEFAULT * 10,
            seed=31137, # training seed
        ),
    ),
]

output_dir = Path("results/charge_eta_bins")
output_dir.mkdir(parents=True, exist_ok=True)

particles = ["pi0", "gamma"]

sim = {}
for p_thrown in energies:
    <<book_sim>>

sim_train = sim
sim = {}
```

```
<<energy_dispatch_def>>
<<train_def>>

def to_x_size(size):
    def to_x(events):
        clusters = find_cluster(events, size=size)
        return np.array(ak.flatten(clusters.egrid, axis=-1))
    return to_x

from xgboost import XGBClassifier

xgboost_classifier = train(
    ak.concatenate([sim_train[p_thrown][0]["pi0"].result() for p_thrown in energies]),
    ak.concatenate([sim_train[p_thrown][0]["gamma"].result() for p_thrown in
    ↪  energies]),
    enable_energy_threshold(to_x_size(5), 50 * MeV),
    XGBClassifier(),
)

sim_settings = [
    dict(
        classifier=enable_energy_threshold(xgboost_classifier, 50 * MeV),
        sim=dict(
            common_sim_params,
            abseta_min=eta,
            abseta_max=eta,
            theta_min=None,
            theta_max=None,
        ),
        label=rf"XGBoost $|\eta| = {eta}$",
        visual=dict(
            color=color,
        ),
    ) for eta, color in zip([0.01, 0.5, 1.], list(mpl.colors.XKCD_COLORS))
]

sim = {}
for p_thrown in energies:
    <<book_sim>>

roc = {}
roc_inv = {}
for p_thrown in energies:
    <<pion_gamma_discrimination>>

    for ix, setting in enumerate(sim_settings):
        def mk_interp(fpr, tpr):
            def interp(eff):
                return np.interp(eff, fpr, tpr)
            return interp
        roc.setdefault(ix, []).append(mk_interp(fpr[ix], tpr[ix]))
        roc_inv.setdefault(ix, []).append(mk_interp(1 - tpr[ix][::-1], 1 -
        ↪  fpr[ix][::-1]))
```

```
for background_level in [1., 5., 10.]:
    for ix, setting in enumerate(sim_settings):
        label = setting["label"]
        visual_params = setting["visual"]
        tpr = np.array([tpr(background_level / 100.) for tpr in roc[ix]])
        plt.plot(energies, tpr * 100, label=f"TPR ($\pi^0$ sensitivity) - {label}",
        ↪  ls="-", **visual_params)
        plt.plot(energies, np.full_like(tpr, background_level), label=rf"FPR$=
        ↪  {background_level:.0f}\%$ (background if $\pi_0:\gamma = 1:1$)", ls=":",
        ↪  **visual_params)

    plt.ylim(0., 100.)
    plt.xlabel(r"$|p_{\mathrm{thrown}}|$, GeV", loc="right")
    plt.ylabel(r"Probability, %", loc="top")
    plt.minorticks_on()
    plt.legend()
    plt.savefig(output_dir / f"pi0_separation_ml_bg{background_level:.0f}.pdf",
    ↪  bbox_inches="tight")
    plt.show()

for background_level in [5., 10., 15., 20.]:
    for ix, setting in enumerate(sim_settings):
        label = setting["label"]
        visual_params = setting["visual"]
        tpr = np.array([tpr(background_level / 100.) for tpr in roc_inv[ix]])
        plt.plot(energies, tpr * 100, label=f"TPR ($\gamma$ sensitivity) - {label}",
        ↪  ls="-", **visual_params)
        plt.plot(energies, np.full_like(tpr, background_level), label=rf"FPR$=
        ↪  {background_level:.0f}\%$ (background if $\pi_0:\gamma = 1:1$)", ls=":",
        ↪  **visual_params)

    plt.ylim(0., 100.)
    plt.xlabel(r"$|p_{\mathrm{thrown}}|$, GeV", loc="right")
    plt.ylabel(r"Probability, %", loc="top")
    plt.minorticks_on()
    plt.legend()
    plt.savefig(output_dir / f"gamma_separation_ml_bg{background_level:.0f}.pdf",
    ↪  bbox_inches="tight")
    plt.show()
```





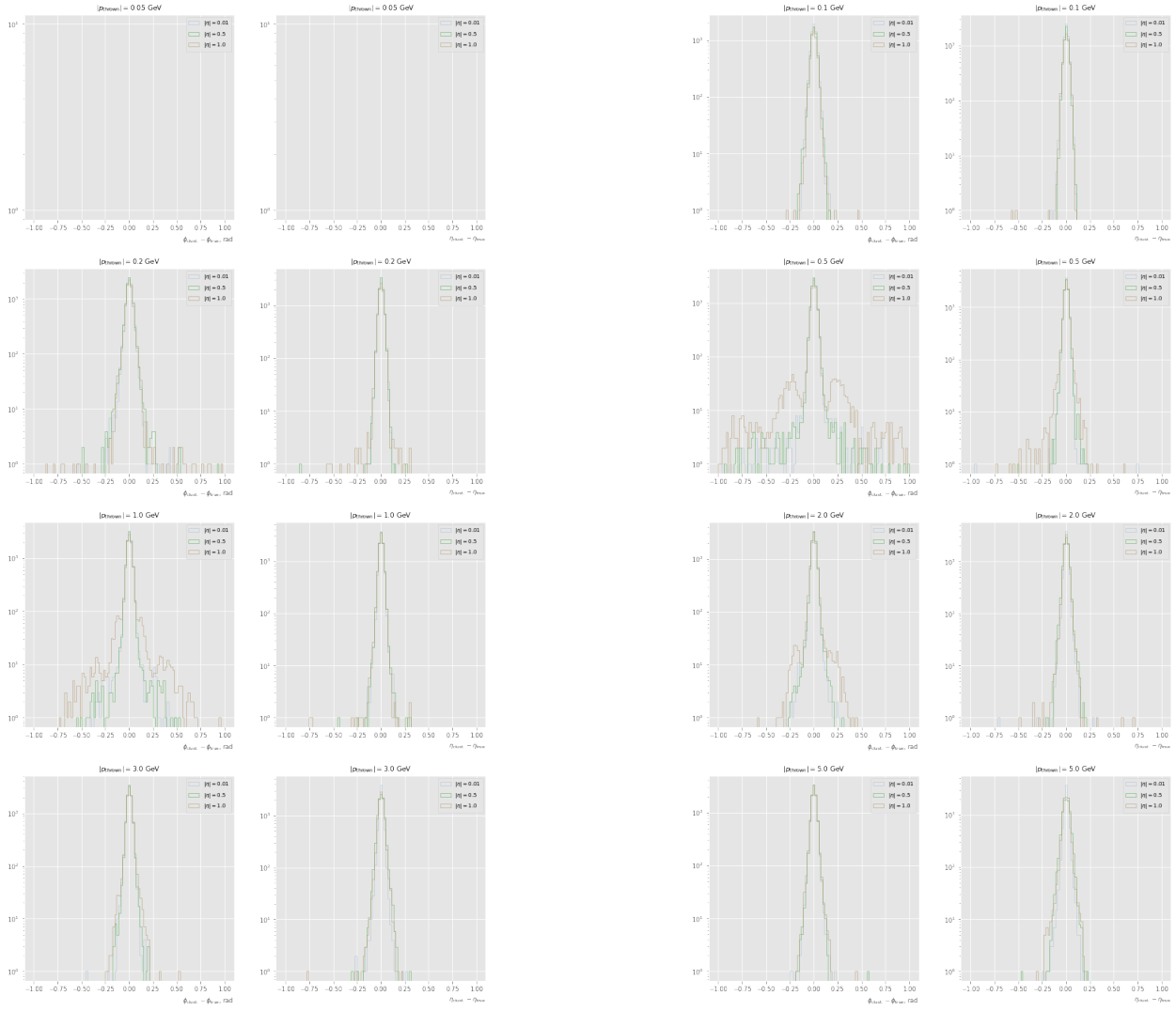
6.6 Angular resolution

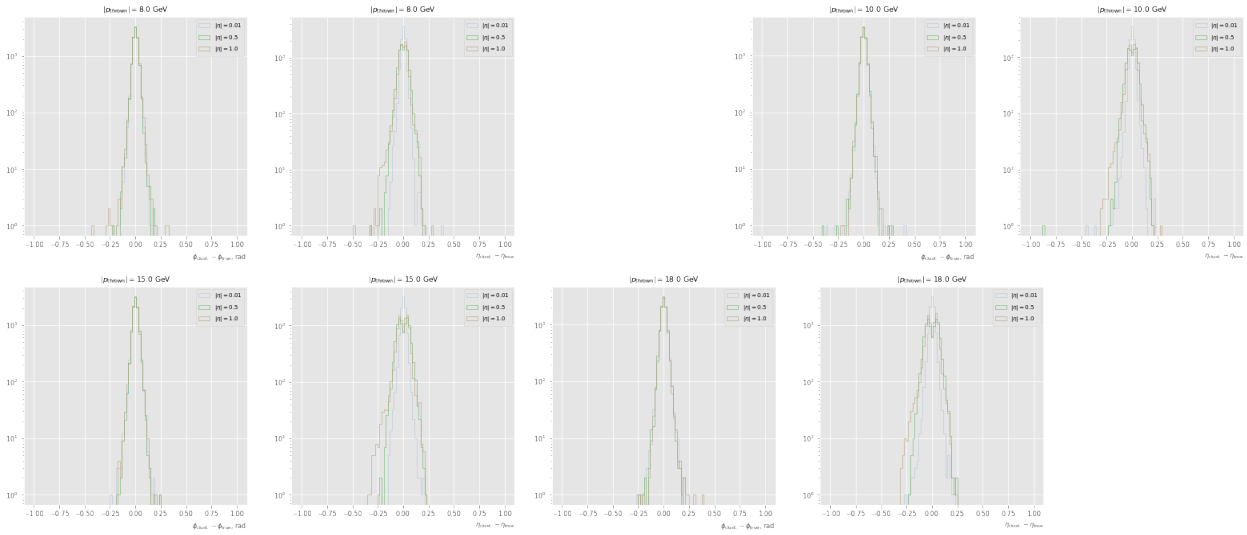
```
<<sim_settings>>

sim_settings = [
    dict(
        sim=dict(
            common_sim_params,
            abseta_min=eta,
            abseta_max=eta,
            theta_min=None,
            theta_max=None,
        ),
        label=rf"$|\eta| = {eta}$",
        visual=dict(
            color=color,
        ),
    ) for eta, color in zip([0.01, 0.5, 1.], list(mpl.colors.XKCD_COLORS))
]

particles = ["gamma"]

output_dir = Path("results/charge_eta_bins")
output_dir.mkdir(parents=True, exist_ok=True)

<<find_island_cluster_def>>

sim = {}
clusters = {}
for p_thrown in energies:
    <<book_sim>>
```

```
    for ix, setting in enumerate(sim_settings):
        for particle in particles:
            clusters.setdefault(p_thrown, {}).setdefault(ix, {})[particle] =
            ↪  client.compute(find_island_cluster(sim[p_thrown][ix][particle]))


<<to_x_with_3momentum_def>>

phi_resolution = {}
eta_resolution = {}

for p_thrown in energies:
    _, ax = plt.subplots(ncols=2, figsize=(11, 5))

    for ix, setting in enumerate(sim_settings):
        label = setting["label"]
        visual_params = setting["visual"]

        _clusters = clusters[p_thrown][ix]["gamma"].result()

        # Select leading energy cluster
        lead_energy_ix = ak.argmax(_clusters["E"], axis=-1)
        has_clusters = ~ak.is_none(lead_energy_ix)
        lead_energy_ix = ak.from_regular(lead_energy_ix[has_clusters,np.newaxis])
        cluster = ak.flatten(_clusters[has_clusters][lead_energy_ix])

        plt.sca(ax[0])
        deltas = (cluster.phi -
        ↪  phi(sim[p_thrown][ix]["gamma"].result()[has_clusters])).to_numpy()
        deltas = np.array([deltas - 2 * np.pi, deltas, deltas + 2 * np.pi])
        deltas, = np.take_along_axis(deltas, np.argmin(np.abs(deltas),
        ↪  axis=0)[np.newaxis,:], axis=0)
        phi_resolution.setdefault(ix, []).append(np.std(deltas))
        plt.hist(deltas,
                 bins=np.linspace(-1., 1., 100),
                 histtype="step", label=label, **visual_params)
        plt.title(rf"$|p_{{\mathrm{{thrown}}}}| = {p_thrown}$ GeV")
        plt.xlabel("$\phi_{\mathrm{clust.}} - \phi_{\mathrm{true}}$, rad", loc="right")

        plt.sca(ax[1])
        deltas = cluster.eta -
        ↪  theta2eta(theta(sim[p_thrown][ix]["gamma"].result()[has_clusters]))
        eta_resolution.setdefault(ix, []).append(np.std(deltas))
        plt.hist(deltas,
                 bins=np.linspace(-1., 1., 100),
                 histtype="step", label=label, **visual_params)
        plt.title(rf"$|p_{{\mathrm{{thrown}}}}| = {p_thrown}$ GeV")
        plt.xlabel("$\eta_{\mathrm{clust.}} - \eta_{\mathrm{true}}$", loc="right")

    plt.sca(ax[0])
    plt.legend()
    plt.yscale("log")
    plt.sca(ax[1])
```

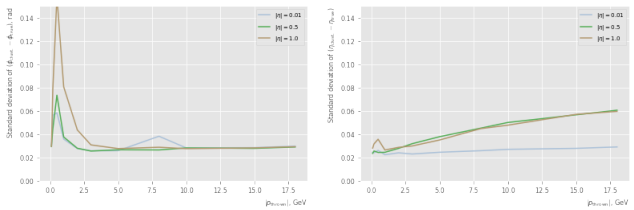
```
    plt.legend()
    plt.yscale("log")
    plt.savefig(output_dir / f"angular_resolution_{p_thrown:.2f}.pdf")
    plt.show()
```

```
/project/rhfate2_uksr/nix/store/gbhqyy6kh56nfp7wcp04xwa9baarp0v5-python3.10-numpy-1.23.1/lib/python3.10/
  ret = _var(a, axis=axis, dtype=dtype, out=out, ddof=ddof,
/project/rhfate2_uksr/nix/store/gbhqyy6kh56nfp7wcp04xwa9baarp0v5-python3.10-numpy-1.23.1/lib/python3.10/
  arrmean = um.true_divide(arrmean, div, out=arrmean, casting='unsafe',
/project/rhfate2_uksr/nix/store/gbhqyy6kh56nfp7wcp04xwa9baarp0v5-python3.10-numpy-1.23.1/lib/python3.10/
  ret = ret.dtype.type(ret / rcount)
/tmp/nix-shell.cLxFME/ipykernel_65508/339375960.py:780: UserWarning: Data has no positive values, and th
  plt.yscale("log")
/tmp/nix-shell.cLxFME/ipykernel_65508/339375960.py:783: UserWarning: Data has no positive values, and th
  plt.yscale("log")
```





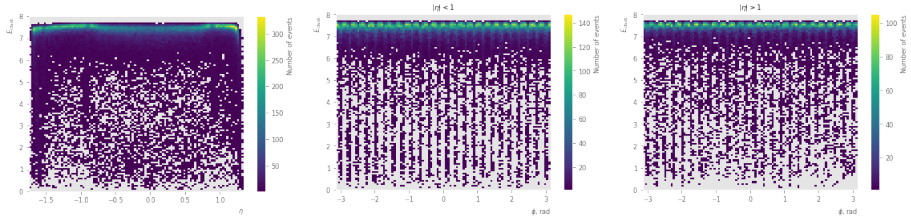
```
for ix, setting in enumerate(sim_settings):
    label = setting["label"]
    visual_params = setting["visual"]
    plt.plot(energies, phi_resolution[ix], label=label, **visual_params)
    plt.xlabel(r"$|p_{\mathrm{thrown}}|$, GeV", loc="right")
    plt.ylabel("Standard deviation of $(\phi_{\mathrm{clust.}} -
    ↪  \phi_{\mathrm{true}})$, rad", loc="top")

plt.ylim(0., 0.15)
plt.legend()
plt.savefig(output_dir / f"angular_resolution_phi.pdf")
plt.show()

for ix, setting in enumerate(sim_settings):
    label = setting["label"]
    visual_params = setting["visual"]
    plt.plot(energies, eta_resolution[ix], label=label, **visual_params)
    plt.xlabel(r"$|p_{\mathrm{thrown}}|$, GeV", loc="right")
    plt.ylabel("Standard deviation of $(\eta_{\mathrm{clust.}} -
    ↪  \eta_{\mathrm{true}})$", loc="top")

plt.ylim(0., 0.15)
plt.legend()
plt.savefig(output_dir / f"angular_resolution_eta.pdf")
plt.show()
```



6.7 Reconstructed cluster energy response to single single electron vs eta and phi

```
output_dir = Path("results/find_island_cluster")
output_dir.mkdir(parents=True, exist_ok=True)

particle = "e-"
p_thrown = 8. * GeV

simu = client.compute(run_ddsim(particle=particle, p_min=p_thrown, p_max=p_thrown,
↪  distribution="pseudorapidity", **dict(common_sim_params, num_events=100000)))
%time simu.result()

%time clusters = client.compute(find_island_cluster(simu)).result()

ix = ak.argmax(clusters["E"], axis=-1)
has_clusters = ~ak.is_none(ix)
ix = ak.from_regular(ix[has_clusters,np.newaxis])

plt.hist2d(ak.flatten(clusters[has_clusters][ix].eta).to_numpy(),
↪  ak.flatten(clusters[has_clusters][ix].mag).to_numpy(), bins=(100, 100), cmin=1)
plt.ylim(0., 8.)
plt.xlabel("$\eta$", loc="right")
plt.ylabel("$E_{\mathrm{clust.}}$", loc="top")
plt.colorbar().set_label("Number of events", loc="top")
plt.savefig(output_dir / "edep_vs_eta.pdf")
plt.show()

for cut_name, title, cond in [
        ("abseta_lt_1", "$|\eta| < 1$",
        ↪  np.abs(ak.flatten(clusters[has_clusters][ix].eta).to_numpy()) < 1.),
        ("abseta_gt_1", "$|\eta| > 1$",
        ↪  np.abs(ak.flatten(clusters[has_clusters][ix].eta).to_numpy()) >= 1.),
]:
    plt.hist2d(ak.flatten(clusters[has_clusters][ix].phi).to_numpy()[cond],
    ↪  ak.flatten(clusters[has_clusters][ix].mag).to_numpy()[cond], bins=(100, 100),
    ↪  cmin=1)
    plt.ylim(0., 8.)
    plt.title(title)
    plt.xlabel("$\phi$, rad", loc="right")
    plt.ylabel("$E_{\mathrm{clust.}}$", loc="top")
    plt.colorbar().set_label("Number of events", loc="top")
    plt.savefig(output_dir / f"edep_vs_phi_{cut_name}.pdf")
    plt.show()
```

```
CPU times: user 3.02 s, sys: 698 ms, total: 3.71 s
Wall time: 4.45 s
CPU times: user 20.6 s, sys: 1.78 s, total: 22.4 s
Wall time: 34.5 s
```


|η| < 1
|η| > 1
Number of events
η
φ, rad