PocketCoffea

Configuration Framework for CMS Analyses based on Coffea

Davide Valsecchi, Matteo Marchegiani, **Felix Zinn**

PyHEP 2025

Overview

- Analysis framework for CMS data in NanoAOD format
- builds a configuration layer on top of Coffea
- common methods/functions already implemented (e.g. object selection, corrections)

Core Components

- Configurator
- Parameters
- Processor

Executing Tasks

- PocketCoffea CLI
- Law tasks

Statistical Analysis Integration

- Processes
- Systematic Uncertainties
- Datacard generation

Previous talk at PyHEP 2023 (Matteo Marchegiani)
PocketCoffea: a configuration layer for CMS analyses with Coffea

GitHub | Documentation

Core Components

Configurator

```py
cfg = Configurator(
  parameters={...},
  datasets={...},
  weights={...},
  variations={...},
  ...
)
```

Parameters

```yaml
HLT_triggers:
  2022_preEE:
    SingleEle:
      - Ele30_WPTight_Gsf
plotting_style:
  labels_mc:
    ttbar: r"$t\bar{t}$"
```

Processor

```py
class MyProcessor(...):
  def count_objects(self):
    ...

  def custom_logic(self):
    ...
```

Configurator

Central component holding all analysis information

```py
cfg = Configurator(
  parameters=<OmegaConf>,
  datasets={},
  skim=[goldenJson, get_HLTsel()],
  preselection=[jet_preselection, get_JetVetoMap],
  weights={"common": {"sf_mu_trigger", ...}, "bysample": {...}},
  variations={"weights": {...}, "shape": {...}},
  variables={...},  # histograms
  columns={...},  # events
  ...)
```

Parameters

Processor

Configurator

Parameters

configuration through YAML files (trigger, preselection, corrections, plotting, ...)
with reasonable defaults in PocketCoffea

```yaml
object_preselection:
  Muon:
    pt: 30
lepton_scale_factors:
  muon_sf:
    JSONfiles:
      '2016_PreVFP':
          file: ${cvmfs:Run2-2016preVFP-UL-NanoAODv9,MUO,muon_Z.json.gz}
plotting_style:
  labels_mc:
    ttbar: r"$t\bar{t}$"
```

Processor

Configurator

Parameters

Processor

Subclass `BaseProcessorABC` for custom analysis logic (hooks provided)

```py
class MyProcessor(BaseProcessorABC):
  def apply_object_preselection(self, variation):
    self.events["ElectronGood"] = electron_selection(
      self.events, self.params
    )
  def define_common_variables_after_presel(self, variation):
    jet_combinations = ak.combinations(
      self.events.JetGood, 2, fields=["jet1", "jet2"]
    )
```

Putting It All Together

Configurator

```py
cfg = Configurator(
  parameters={...},
  datasets={...},
  weights={...},
  variations={...},
  ...
)
```

Parameters

```yaml
HLT_triggers:
  2022_preEE:
    SingleEle:
      - Ele30_WPTight_Gsf
plotting_style:
  labels_mc:
    ttbar: r"$t\bar{t}$"
```

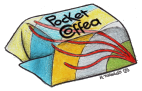
Processor

```py
class MyProcessor(...):
  def count_objects(self):
    ...

  def custom_logic(self):
    ...
```

- Define analysis configuration in the **Configurator**
- Specify parameters in **YAML files**
- Implement custom analysis logic in your **Processor**

Running Tasks in PocketCoffea



Core Components

- Configurator
- Parameters
- Processor

Executing Tasks

- PocketCoffea CLI
- Law tasks

Statistical Analysis Integration

- Processes
- Systematic Uncertainties
- Datacard generation

PocketCoffea CLI: Overview

- Provides common analysis tasks and utilities
- Enables dataset building, running, plotting, and more

```
❯ pocket-coffea --help
  Usage: pocket-coffea [OPTIONS] COMMAND [ARGS]...
  ...
  Commands:
    build-datasets         Build dataset fileset in json format
    dataset-discovery-cli  CLI for interactive dataset discovery.
    make-config            Create a template for an analysis configuration...
    make-plots
    run                    Run an analysis on NanoAOD files using...
    ...
```

PocketCoffea CLI: Highlights

make-config

- Guided template generator
- Asks questions, creates starter config
- Jumpstart your analysis setup

dataset-discovery-cli

- Interactive CLI to query CMS datasets
- Select datasets and grid sites for processing
- Save as dataset config

run

- Run analysis config on datasets
- locally or on cluster
- Supports multiple executors (futures, parsl@RWTH, ...)

Custom Executors


```py
from dask_jobqueue import HTCondorCluster
from distributed import Client

class MyExecutor(ExecutorFactoryABC):
  def setup(self):
    self.dask_cluster = HTCondorCluster(
            cores  = self.run_options['cores-per-worker'],
            memory = self.run_options['mem-per-worker'],
            disk = self.run_options.get('disk-per-worker', "2GB"),
            job_script_prologue = self.get_worker_env(),
    )
    self.dask_client = Client(self.dask_cluster)
  def get(self):
    return coffea_processor.dask_executor(client=self.dask_client)
```

Predefined Executors

Site	Supported executors
lxplus	dask
swan	dask
`T3_CH_PSI`	dask
DESY NAF	parsl
RWTH Aachen LX-Cluster	parsl, dask
RWTH CLAIX	parsl, dask
Purdue Analysis Facility	dask
INFN Analysis Facility	dask
Brown brux20 cluster	dask
Brown CCV Oscar	dask
Maryland rubin cluster	dask, condor

PocketCoffea CLI: Typical Workflow

1. Define datasets
   `pocket-coffea build-datasets`
2. Write config & processor
   `pocket-coffea run`
3. Create plots
   `pocket-coffea make-plots`

→ Do this with law tasks

Workflow Tasks with Law

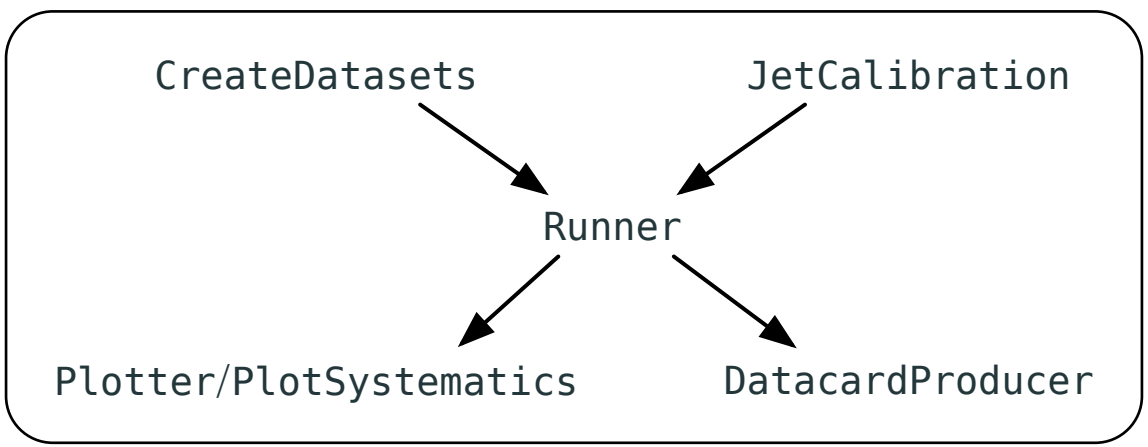
- law is workflow management tool (https://law.readthedocs.io/)
- structured representation of analysis workflow
- dependencies are run if output not existent
- automatic handling of task output (e.g. different versions, removal, file transfer)

```
❯ law run Plotter --version v1 --cfg analysis_config.py --print-status -1

0 > Plotter  --> absent
|
└──1 > Runner  --> absent
   |
   ├──2 > CreateDatasets  --> absent
   |
   └──2 > JetCalibration  --> existent
```

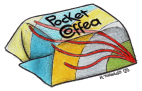
Workflow Tasks with Law

current implementation in PocketCoffea



- extensible through custom tasks (e.g. neural network training, statistical analysis, ...)
- uses same functions as CLI → User Preference to use either

Statistical Analysis Integration



Core Components

- Configurator
- Parameters
- Processor

Executing Tasks

- PocketCoffea CLI
- Law tasks

Statistical Analysis Integration

- Processes
- Systematic Uncertainties
- Datacard generation

Statistical Analysis Integration


```
'variables': ▼{
  'JetGood_pt': ▼defaultdict({
    'TTH_Hto2B': ▼{
      'TTH_Hto2B_2022_preEE': ▼# <hist.hist.Hist object at 0x72a829979410>
        Hist(
          StrCategory(['inclusive'], name='cat', label='Category'),
          StrCategory(['nominal', 'pileupDown', 'pileupUp'], name='variation', label='Variation'),
          Variable(array([  0.,   5.,  10.,  15.,  20.,  25.,  30.,  35.,  40.,  45.,  50.,
```

→ export to use with statistics tools (samples, histograms, uncertainties, ...)

Statistical Analysis Integration

Process

```py
ttbar = MCProcess(
  name="ttbar",
  samples=["TTTo2L2Nu"],
  years=["2022_preEE"],
  is_signal=False,
)
```

SystematicUncertainty

```py
lumi = SystematicUncertainty(
  name="lumi",
  typ="lnN",
  processes={"ttbar": 1.025},
  years=["2022_preEE"],
)
```

Datacard

```py
datacard = Datacard(
  mc_processes=[ttbar],
  systematics=[lumi],
  histograms=histos,
  years=["2022_preEE"],
)
```

Process

- Group samples into processes
- Define MC and Data processes

```py
@dataclass
class Process:
  name: str
  samples: Iterable
  label: str = None
  is_data: bool = field(init=False)
```

SystematicUncertainty

Datacard

Statistical Analysis Integration

Process

- Group samples into processes
- Define MC and Data processes

```py
@dataclass(kw_only=True)
class MCProcess(Process):
  is_signal: bool
  years: Iterable
  has_rateParam: bool = False

  def __post_init__(self):
    self.is_data = False
    super().__post_init__()
```

```py
@dataclass(kw_only=True)
class DataProcess(Process):
  years: Iterable

  def __post_init__(self):
    self.is_data = True
    super().__post_init__()

```

SystematicUncertainty

Datacard

Statistical Analysis Integration

Process

- Group samples into processes
- Define MC and Data processes

```py
mc_process = MCProcess(
  name="ttbar",
  samples=["TTTo2L2Nu",
  "TTToSemiLeptonic"],
  years=["2022_preEE", "2022_postEE"],
  is_signal=False,
  has_rateParam=True,
)
```

```py
data_process = DataProcess(
  name="data_obs",
  samples=["DATA_SingleMuon",
  "DATA_SingleElectron"],
  years=["2022_preEE", "2022_postEE"],
)
```

SystematicUncertainty

Datacard

Process

SystematicUncertainty

Define uncertainties that apply to processes/years

```py
@dataclass
class SystematicUncertainty:
  name: str
  typ: str
  processes: list[str] | tuple[str] | dict[str, float]
  years: list[str] | tuple[str]
  value: float | tuple[float] = None
  datacard_name: str = None
```

Datacard

Process

SystematicUncertainty

Define uncertainties that apply to processes/years

```py
syst = SystematicUncertainty(
  name="syst",
  typ="lnN",
  processes={
    "ttbar": 1.025,
    "dy": (0.9, 1.1),
  },
  years=["2022_preEE", "2022_postEE"],
)
```

Datacard

Process

SystematicUncertainty

Datacard

- Create combine-ready datacards from PocketCoffea output (CMS Combine)
- Per-variable, per-category outputs → merge with `combineCards.py`

```py
datacard = Datacard(
  mc_processes=mc_processes,
  systematics=systematics,
  histograms=histograms,
  years=["2022_preEE", "2022_postEE"],
  category="category_name",
  data_processes=data_processes,
  ...
)
```

Putting It All Together

Process

```py
ttbar = MCProcess(
  name="ttbar",
  samples=["TTTo2L2Nu"],
  years=["2022_preEE"],
  is_signal=False,
)
```

SystematicUncertainty

```py
lumi = SystematicUncertainty(
  name="lumi",
  typ="lnN",
  processes={"ttbar": 1.025},
  years=["2022_preEE"],
)
```

Datacard

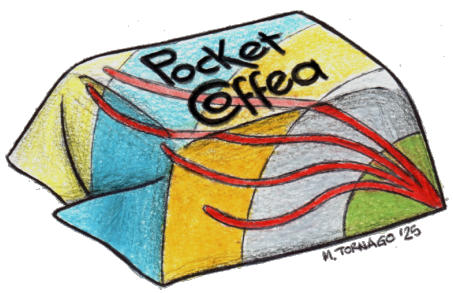
```py
datacard = Datacard(
  mc_processes=[ttbar],
  systematics=[lumi],
  histograms=histos,
  years=["2022_preEE"],
)
```

1. Define processes (MC and Data) for your analysis
2. Specify systematic uncertainties and which processes they affect
3. Create datacards per category/variable using histograms from PocketCoffea output
4. Merge datacards with `combineCards.py` and run statistical inference with `combine`

Summary

Key Features

- **Declarative configuration** via `Configurator` and YAML files
- **PocketCoffea CLI** for common analysis tasks
- **Workflow tasks via law**
- **Statistical analysis integration**



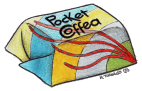
Recent Updates

- CMS CAT Versioning scheme support (choose correction tags)
- Coffea v2025 with awkward virtual arrays (test branch)

GitHub

Documentation

Statistical Analysis Integration (backup)

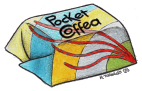


```py
from pocket_coffea.utils.stat import MCProcess, DataProcess

# example of a monte carlo process
mc_process = MCProcess(
    name="ttbar",
    samples=["TTTo2L2Nu", "TTToSemiLeptonic"],
    years=["2022_preEE", "2022_postEE"],
    is_signal=False,
    has_rateParam=True,
)

# example of a data process
data_process = DataProcess(
    name="data_obs",
    samples=["DATA_SingleMuon", "DATA_SingleElectron"],
    years=["2022_preEE", "2022_postEE"],
)
```

Statistical Analysis Integration (backup)

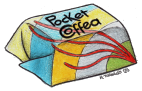


```py
from pocket_coffea.utils.stat import SystematicUncertainty

syst = SystematicUncertainty(
    name="syst",
    type="lnN",
    processes=["ttbar"],
    value=1.025,
    years=["2022_preEE", "2022_postEE"],
)

# or with asymmetric uncertainties
syst = SystematicUncertainty(
    name="syst",
    type="lnN",
    processes=["ttbar"],
    value=(0.995, 1.015),  # asymmetric log-normal
    years=["2022_preEE", "2022_postEE"],
)
```

Statistical Analysis Integration (backup)



```py
# or with different values for different processes
syst = SystematicUncertainty(
    name="syst",
    type="lnN",
    processes={
        "ttbar": 1.025,
        "dy": (0.9, 1.1),
    },
    years=["2022_preEE", "2022_postEE"],
)
```

Statistical Analysis Integration (backup)

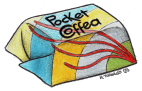
```py
from pocket_coffea.utils.stat import Datacard

# load coffea output
output = load("output.coffea")

# define Processes and Systematics
mc_processes = Processes(
    [
        MCProcess(name="ttbar", ...),
        MCProcess(name="dy", ...)
    ]
)

data_processes = Processes(
    [
        DataProcess(name="data_obs", ...),
    ]
)
```

Statistical Analysis Integration (backup)




```py
systematics = Systematics(
    [
        SystematicUncertainty(name="syst1", ...),
        SystematicUncertainty(name="syst2", ...),
    ]
)

datacard = Datacard(
    histograms=histograms,
    datasets_metadata=output["datasets_metadata"],
    cutflow=output["cutflow"],
    years=["2022_preEE", "2022_postEE"],
    mc_processes=mc_processes,
    systematics=systematics,
    category="category_name",
    data_processes=data_processes,
)
```