Analysis Grand Challenge Related Tests

Prompt

> Implement a complete H->ZZ->4l analysis pipeline in the template file `analysis.py`. Make sure `analysis.py` is PEP 723 compliant, and use the `uv` tools to run and test it. Use the appropriate `agc-skills` skills for further instructions.
>
> Create a new code block (separated by an empty line) for each major step. The pipeline consists of four main stages, to be implemented in order:
>
> Dataset retrieval - Retrieve the input columns from the input_files dictionary (Data, Background Z ttbar, Background ZZ*, Signal). Tree name: mini. Event selection: exactly four leptons (lep_n == 4). Columns needed downstream: lepton pt, eta, phi, energy, charge, type, mcWeight, and the scale factors scaleFactor_ELE, scaleFactor_MUON, scaleFactor_LepTRIGGER, scaleFactor_PILEUP. Compute a scale factor variation systematic at query stage (product of the four scale factors above, scaled up by 1.1 and down by 0.9). Support both a func_adl (FuncADL_Uproot) query and an UprootRaw alternative, selectable via a USE_SERVICEX_UPROOT_RAW flag, and respect an IGNORE_CACHE flag.
>
> Event processing - Process the retrieved columns to build the discriminating variable m4l (invariant mass of the four-lepton system). Select events with exactly two same-flavor opposite-charge lepton pairs (electrons or muons). Produce histograms of m4l for one signal region, for each sample (Data, Signal MC, Background ZZ*, Background Z ttbar), including variations for the following systematics: scale factor variation, 4l mass variation, and ZZ normalization. Also account for a signal normalization factor.
>
> Plotting - Plot the m4l histograms for the signal region, showing all samples and the systematic variations listed above. Create a figures directory as a sub-directory of the working sandbox directory if one does not already exist, and save each plot as both a .pdf and a .png file in that directory.
>
> Statistical analysis - Using the histograms produced above, build a cabinetry configuration, collect and postprocess the templates, build a workspace, and perform the fit. The parameter of interest is the signal normalization. Produce and save (as .pdf and .png in the figures

> directory) a pull plot (name: pull_plot.pdf/.png), a correlation matrix plot (name: matrix.pdf/.png), and a post-fit data/model comparison plot (name: model-comp.pdf/.png) for the signal region with a meaningful x-axis label. Finally, write a results.json file containing the POI value and error, formatted as:
>
> {
>
> “poi_val” : x,
>
> “poi_err” : y
>
> }
>
> After each stage, run the Analysis.py file and capture any error output. If there is an error, examine the message and correct it to the best of your ability, then re-run until the file executes without errors and the output looks as expected.

Final response

> Implemented the complete pipeline in analysis.py with PEP 723 dependencies and distinct notebook-style blocks for every stage. Added the cabinetry model in config.yml.
>
> Verified with `uv`:
>
> - UprootRaw and FuncADL_Uproot paths both pass.
> - Full pipeline exits successfully.
> - Ruff lint and formatting checks pass.
> - Selected data: 369 events.
> - Nominal MC yield: 288.221.
> - Fitted signal normalization: `1.1368 ± 0.7022`.
>
> Outputs include:
>
> - results.json
> - workspace.json
> - histograms.root
> - pull_plot.png
> - matrix.png
> - model-comp.png
>
> The AGC skills guided the ServiceX local-cache delivery, Coffea/BaseSchema processing, systematic naming, ROOT-template export, and cabinetry collect/postprocess workflow. Cross-section constants follow the IRIS-HEP HZZ reference implementation.

Skills and marketplaces

- Marketplaces:
  - `https://github.com/gordonwatts/agc_marketplace`
- Plugins:
  - `agc-skills@agc_marketplace`
- Skill directories:
  - `plugins/cache/agc_marketplace/agc-skills/0.1.0/skills/cabinetry-analysis`

- `plugins/cache/agc_marketplace/agc-skills/0.1.0/skills/coffea-processing`
- `plugins/cache/agc_marketplace/agc-skills/0.1.0/skills/corrections-systematics`
- `plugins/cache/agc_marketplace/agc-skills/0.1.0/skills/dataset-retrieval`
- `plugins/cache/agc_marketplace/agc-skills/0.1.0/skills/hist-plotter`

Run summary

Field	Value
Run date	2026-09-07
Started	07:56:54
Finished	08:20:37
Total duration	0h 23m 42s
Codex execution	0h 23m 29s
Status	succeeded
Exit code	0
Distribution	Ubuntu
Codex version	codex-cli 0.153.4
Workspace	/home/gwatts/llm-testing/test-wsl2-llm-4ZkJHL6k/workspace
Workspace retained	True

Phase timing

Phase	Started	Finished	Duration
preflight	07:56:54	07:56:58	4s
workspace_creation	07:56:58	07:56:58	0s
input_transfer	07:56:58	07:57:01	3s
codex_home_setup	07:57:01	07:57:01	0s
plugin_installation	07:57:01	07:57:03	1s
codex_execution	07:57:03	08:20:31	23m 29s
workspace_inventory	08:20:31	08:20:34	2s
copy_back	08:20:34	08:20:36	2s

Phase	Started	Finished	Duration
session_trace_collection	08:20:36	08:20:36	0s

Token usage and cost

- Pricing file: `bundled:model-pricing.yaml`
- Currency: `USD`
- Token counts come from the `usage` object on each captured `turn.completed` event in Codex stdout JSONL; the configured model is used because those events do not report a model name.
- `input_tokens` is the total input-token count, including the `cached_input_tokens` subset. Uncached input is calculated as `input_tokens - cached_input_tokens`; cached input was served from the prompt cache and is priced at its separate cached-input rate.
- Output tokens are reported separately. Costs use the uncached-input rate for uncached input, the cached-input rate for cached input, and the output rate for output tokens.

Model	Attribution	Input	Cached input	Output	Reasoning output	USD total
gpt-5.6-sol	inferred - model not directly reported	6,183,522	6,052,608	24,645	6,579	$3.44
Aggregate						**$3.44**

Invocation

> C:\Users\gordo\AppData\Local\uv\cache\archive-v0\ORp9JIYJhVbwSU8c\Scripts\test-wsl2-llm template run .\agc-tests.yaml --model gpt-5.6-sol:medium

Codex command

```
# TEST_WSL2_LLM_ARG_* values are base64-encoded WSL transport arguments (such as paths a
wsl.exe -d Ubuntu -- env TEST_WSL2_LLM_ARG_0=L2hvbWUvZ3dhdHRzL2xsbS10ZXN0aW5nL3Rlc3Qtd3N
```

▶ Workspace inventory

Copied-back files

agc-q1-full.analysis.py

- Source: `analysis.py`
- Size: 19,152 bytes

Text contents (first 10 lines visible; scroll for the rest):

```
#!/usr/bin/env -S uv run
# /// script
# requires-python = ">=3.11,<3.13"
# dependencies = [
#   "awkward>=2.8",
#   "cabinetry>=0.6",
#   "coffea>=2025.7",
#   "hist>=2.8",
#   "matplotlib>=3.10",
#   "mplhep>=0.4",
#   "numpy>=2.0",
#   "pyhf>=0.7",
#   "servicex>=3.0",
#   "uproot>=5.6",
#   "vector>=1.6",
# ]
# ///

"""Complete ATLAS Open Data H -> ZZ* -> 4l analysis pipeline.

Run with ``uv run analysis.py``. Set ``HZZ_STOP_AFTER`` to one of
``retrieval``, ``processing``, or ``plotting`` to execute only through that
stage. ServiceX's UprootRaw query is the default; set
``USE_SERVICEX_UPROOT_RAW=0`` to exercise the FuncADL_Uproot query instead.
"""

# %% [markdown]
# # H -> ZZ* -> 4l analysis configuration and imports

# %%
import json
import os
import re
import time
from pathlib import Path

import awkward as ak
import cabinetry
import hist
import matplotlib.pyplot as plt
import mplhep
import numpy as np
import servicex
import uproot
import vector
```

```
from coffea import processor
from coffea.nanoevents.schemas.base import BaseSchema

vector.register_awkward()

WORK_DIR = Path(__file__).resolve().parent
os.chdir(WORK_DIR)
FIGURES_DIR = WORK_DIR / "figures"
HISTOGRAM_FILE = WORK_DIR / "histograms.root"
CONFIG_FILE = WORK_DIR / "config.yml"
WORKSPACE_FILE = WORK_DIR / "workspace.json"
RESULTS_FILE = WORK_DIR / "results.json"
SERVICEX_CACHE_DIR = WORK_DIR / ".servicex-cache"

CHUNKSIZE = 500_000
NUM_CORES = 4
IGNORE_CACHE = os.getenv("IGNORE_CACHE", "0").lower() in {"1", "true", "yes"}
USE_SERVICEX_UPROOT_RAW = os.getenv("USE_SERVICEX_UPROOT_RAW", "1").lower() not in {
    "0",
    "false",
    "no",
}
STOP_AFTER = os.getenv("HZZ_STOP_AFTER", "").lower()

TREE_NAME = "mini"
BIN_EDGE_LOW = 80.0
BIN_EDGE_HIGH = 250.0
NUM_BINS = 34
LUMINOSITY_PB = 10_000.0
SIGNAL_NORMALIZATION = 1.0

DATA = "Data"
Z_TTBAR = "Background Z, ttbar"
ZZSTAR = "Background ZZ*"
SIGNAL = "Signal"
MC_SAMPLES = (Z_TTBAR, ZZSTAR, SIGNAL)
VARIATIONS = (
    "nominal",
    "scale_factor_up",
    "scale_factor_down",
    "m4l_up",
    "m4l_down",
    "zz_norm_up",
    "zz_norm_down",
)
```

```
prefix = "http://xrootd-local.unl.edu:1094//store/user/AGC/ATLAS_HZZ/"
input_files = {
    DATA: [prefix + f"data_{period}.4lep.root" for period in "ABCD"],
    Z_TTBAR: [
        prefix + "mc_361106.Zee.4lep.root",
        prefix + "mc_361107.Zmumu.4lep.root",
        prefix + "mc_410000.ttbar_lep.4lep.root",
    ],
    ZZSTAR: [prefix + "mc_363490.llll.4lep.root"],
    SIGNAL: [
        prefix + "mc_345060.ggH125_ZZ4lep.4lep.root",
        prefix + "mc_344235.VBFH125_ZZ4lep.4lep.root",
        prefix + "mc_341964.WH125_ZZ4lep.4lep.root",
        prefix + "mc_341947.ZH125_ZZ4lep.4lep.root",
    ],
}

# Values published with the ATLAS Open Data HZZ example: cross-section in pb,
# generated sum of weights, and reduction efficiency.
SAMPLE_INFO = {
    "Zee": (1950.5295, 150277594200.0, 1.0),
    "Zmumu": (1950.6321, 147334691090.0, 1.0),
    "ttbar_lep": (452.693559, 49386600.0, 1.0),
    "llll": (1.2578, 7538705.8077, 1.0),
    "ggH125_ZZ4lep": (0.0060239, 27881776.6536, 1.0),
    "VBFH125_ZZ4lep": (0.0004633012, 3680490.83243, 1.0),
    "WH125_ZZ4lep": (0.0003769, 149400.0, 1.0),
    "ZH125_ZZ4lep": (0.0000021424784, 150000.0, 1.0),
}


# %% [markdown]
# ## 1. Dataset retrieval
#
# Select exactly four-lepton events and only the columns used downstream. The
# nominal scale factor and its +/-10% variations are calculated by ServiceX.


# %%
def get_lepton_query():
    """Build the FuncADL_Uproot query for the flat ``mini`` tree."""
    from servicex import query as q

    return (
        q.FuncADL_Uproot()
        .FromTree(TREE_NAME)
```

```
        .Where(lambda event: event.lep_n == 4)
        .Select(
            lambda event: {
                "lep_pt": event.lep_pt,
                "lep_eta": event.lep_eta,
                "lep_phi": event.lep_phi,
                "lep_energy": event.lep_E,
                "lep_charge": event.lep_charge,
                "lep_type": event.lep_type,
                "mcWeight": event.mcWeight,
                "scaleFactor_ELE": event.scaleFactor_ELE,
                "scaleFactor_MUON": event.scaleFactor_MUON,
                "scaleFactor_LepTRIGGER": event.scaleFactor_LepTRIGGER,
                "scaleFactor_PILEUP": event.scaleFactor_PILEUP,
                "scaleFactor": event.scaleFactor_ELE
                * event.scaleFactor_MUON
                * event.scaleFactor_LepTRIGGER
                * event.scaleFactor_PILEUP,
                "scaleFactorUP": event.scaleFactor_ELE
                * event.scaleFactor_MUON
                * event.scaleFactor_LepTRIGGER
                * event.scaleFactor_PILEUP
                * 1.1,
                "scaleFactorDOWN": event.scaleFactor_ELE
                * event.scaleFactor_MUON
                * event.scaleFactor_LepTRIGGER
                * event.scaleFactor_PILEUP
                * 0.9,
            }
        )
    )


def get_lepton_query_uproot_raw():
    """Build the equivalent UprootRaw query, including the event-level cut."""
    from servicex import query as q

    scale_factor = (
        "scaleFactor_ELE*scaleFactor_MUON*scaleFactor_LepTRIGGER*scaleFactor_PILEUP"
    )
    return q.UprootRaw(
        [
            {
                "treename": TREE_NAME,
                "expressions": [
                    "lep_pt",
```

```
                    "lep_eta",
                    "lep_phi",
                    "lep_energy",
                    "lep_charge",
                    "lep_type",
                    "mcWeight",
                    "scaleFactor_ELE",
                    "scaleFactor_MUON",
                    "scaleFactor_LepTRIGGER",
                    "scaleFactor_PILEUP",
                    "scaleFactor",
                    "scaleFactorUP",
                    "scaleFactorDOWN",
                ],
                "aliases": {
                    "lep_energy": "lep_E",
                    "scaleFactor": scale_factor,
                    "scaleFactorUP": f"({scale_factor})*1.1",
                    "scaleFactorDOWN": f"({scale_factor})*0.9",
                },
                "cut": "lep_n == 4",
            }
        ]
    )


query = get_lepton_query_uproot_raw() if USE_SERVICEX_UPROOT_RAW else get_lepton_query()
bundle = {
    "General": {"Delivery": "LocalCache"},
    "Sample": [
        {
            "Name": name,
            "Query": query,
            "Dataset": servicex.dataset.FileList(paths),
            "IgnoreLocalCache": IGNORE_CACHE,
        }
        for name, paths in input_files.items()
    ],
}

retrieval_start = time.monotonic()
retrieved = servicex.deliver(bundle, cache_dir=str(SERVICEX_CACHE_DIR))
fileset = {
    name: {"files": list(retrieved[name]), "metadata": {"dataset_name": name}}
    for name in retrieved
}
```

```
print(
    f"Dataset retrieval completed in {time.monotonic() - retrieval_start:.2f} s: "
    + ", ".join(
        f"{name}={len(entry['files'])} files" for name, entry in fileset.items()
    )
)

if STOP_AFTER == "retrieval":
    raise SystemExit(0)


# %% [markdown]
# ## 2. Event processing
#
# Form four-vectors, require two disjoint same-flavor opposite-charge pairs,
# calculate m4l, normalize MC to 10 fb^-1, and fill all requested variations.


# %%
def lepton_pair_filter(lep_charge, lep_type):
    """Return events that can be partitioned into exactly two e/mu SFOS pairs."""
    abs_type = abs(lep_type)
    is_electron = abs_type == 11
    is_muon = abs_type == 13
    positive = lep_charge > 0
    negative = lep_charge < 0

    e_positive = ak.sum(is_electron & positive, axis=1)
    e_negative = ak.sum(is_electron & negative, axis=1)
    mu_positive = ak.sum(is_muon & positive, axis=1)
    mu_negative = ak.sum(is_muon & negative, axis=1)
    supported_flavors = ak.all(is_electron | is_muon, axis=1)
    return (
        supported_flavors
        & (e_positive == e_negative)
        & (mu_positive == mu_negative)
        & (e_positive + mu_positive == 2)
    )


def sample_name_from_filename(filename: str) -> str:
    """Extract the ATLAS sample identifier from an input or delivered URL."""
    match = re.search(r"mc_\d+\.([^.]+)\.4lep", filename)
    if match is None:
        raise ValueError(f"Could not identify MC sample from filename: {filename}")
    return match.group(1)
```

```
def cross_section_weight(filename: str) -> float:
    """Return luminosity * cross-section / generated sum-of-weights."""
    sample_name = sample_name_from_filename(filename)
    cross_section, sum_weights, reduction_efficiency = SAMPLE_INFO[sample_name]
    return LUMINOSITY_PB * cross_section / (sum_weights * reduction_efficiency)


def empty_region_histograms():
    """Construct mergeable weighted histograms for the signal region."""
    observable_axis = hist.axis.Regular(
        NUM_BINS,
        BIN_EDGE_LOW,
        BIN_EDGE_HIGH,
        name="m4l",
        label=r"$m_{4\ell}$ [GeV]",
    )
    data_histogram = hist.Hist(observable_axis, storage=hist.storage.Weight())
    mc_histogram = hist.Hist(
        observable_axis,
        hist.axis.StrCategory(MC_SAMPLES, name="process"),
        hist.axis.StrCategory(VARIATIONS, name="variation"),
        storage=hist.storage.Weight(),
    )
    return {"data": data_histogram, "mc": mc_histogram}


class HZZAnalysis(processor.ProcessorABC):
    """Coffea processor for the H -> ZZ* -> 4l signal region."""

    def process(self, events):
        # Worker processes need the vector behavior registration as well.
        vector.register_awkward()
        dataset = events.metadata["dataset_name"]
        selected = events[lepton_pair_filter(events.lep_charge, events.lep_type)]
        leptons = ak.zip(
            {
                "pt": selected.lep_pt,
                "eta": selected.lep_eta,
                "phi": selected.lep_phi,
                "energy": selected.lep_energy,
            },
            with_name="Momentum4D",
        )
        four_lepton = leptons[:, 0] + leptons[:, 1] + leptons[:, 2] + leptons[:, 3]
```

```
        m4l = ak.to_numpy(four_lepton.mass / 1000.0)
        output = {"signal_region": empty_region_histograms()}

        if dataset == DATA:
            output["signal_region"]["data"].fill(m4l=m4l)
            return output

        normalization = cross_section_weight(events.metadata["filename"])
        if dataset == SIGNAL:
            normalization *= SIGNAL_NORMALIZATION
        basic_weight = normalization * ak.to_numpy(selected.mcWeight)
        nominal_weight = basic_weight * ak.to_numpy(selected.scaleFactor)
        sf_up_weight = basic_weight * ak.to_numpy(selected.scaleFactorUP)
        sf_down_weight = basic_weight * ak.to_numpy(selected.scaleFactorDOWN)
        mc_histogram = output["signal_region"]["mc"]

        mc_histogram.fill(
            m4l=m4l, process=dataset, variation="nominal", weight=nominal_weight
        )
        mc_histogram.fill(
            m4l=m4l,
            process=dataset,
            variation="scale_factor_up",
            weight=sf_up_weight,
        )
        mc_histogram.fill(
            m4l=m4l,
            process=dataset,
            variation="scale_factor_down",
            weight=sf_down_weight,
        )
        mc_histogram.fill(
            m4l=m4l * 1.01,
            process=dataset,
            variation="m4l_up",
            weight=nominal_weight,
        )
        mc_histogram.fill(
            m4l=m4l * 0.99,
            process=dataset,
            variation="m4l_down",
            weight=nominal_weight,
        )
        if dataset == ZZSTAR:
            mc_histogram.fill(
                m4l=m4l,
```

```
                process=dataset,
                variation="zz_norm_up",
                weight=nominal_weight * 1.1,
            )
            mc_histogram.fill(
                m4l=m4l,
                process=dataset,
                variation="zz_norm_down",
                weight=nominal_weight * 0.9,
            )
        return output

    def postprocess(self, accumulator):
        return accumulator


processing_start = time.monotonic()
executor = processor.FuturesExecutor(workers=NUM_CORES)
runner = processor.Runner(
    executor=executor,
    savemetrics=True,
    metadata_cache={},
    chunksize=CHUNKSIZE,
    schema=BaseSchema,
)
all_histograms, metrics = runner(
    fileset,
    HZZAnalysis(),
    treename=TREE_NAME if USE_SERVICEX_UPROOT_RAW else "servicex",
)
signal_region = all_histograms["signal_region"]
print(
    f"Event processing completed in {time.monotonic() - processing_start:.2f} s; "
    f"selected data={signal_region['data'].sum().value:.0f}, "
    f"nominal MC={signal_region['mc'][:, :, 'nominal'].sum().value:.3f}"
)

if STOP_AFTER == "processing":
    raise SystemExit(0)


# %% [markdown]
# ## 3. Plotting and template export
#
# Plot the signal-region data and nominal stacked MC, then compare every
# systematic up/down pair. Every figure is written as both PDF and PNG.
```

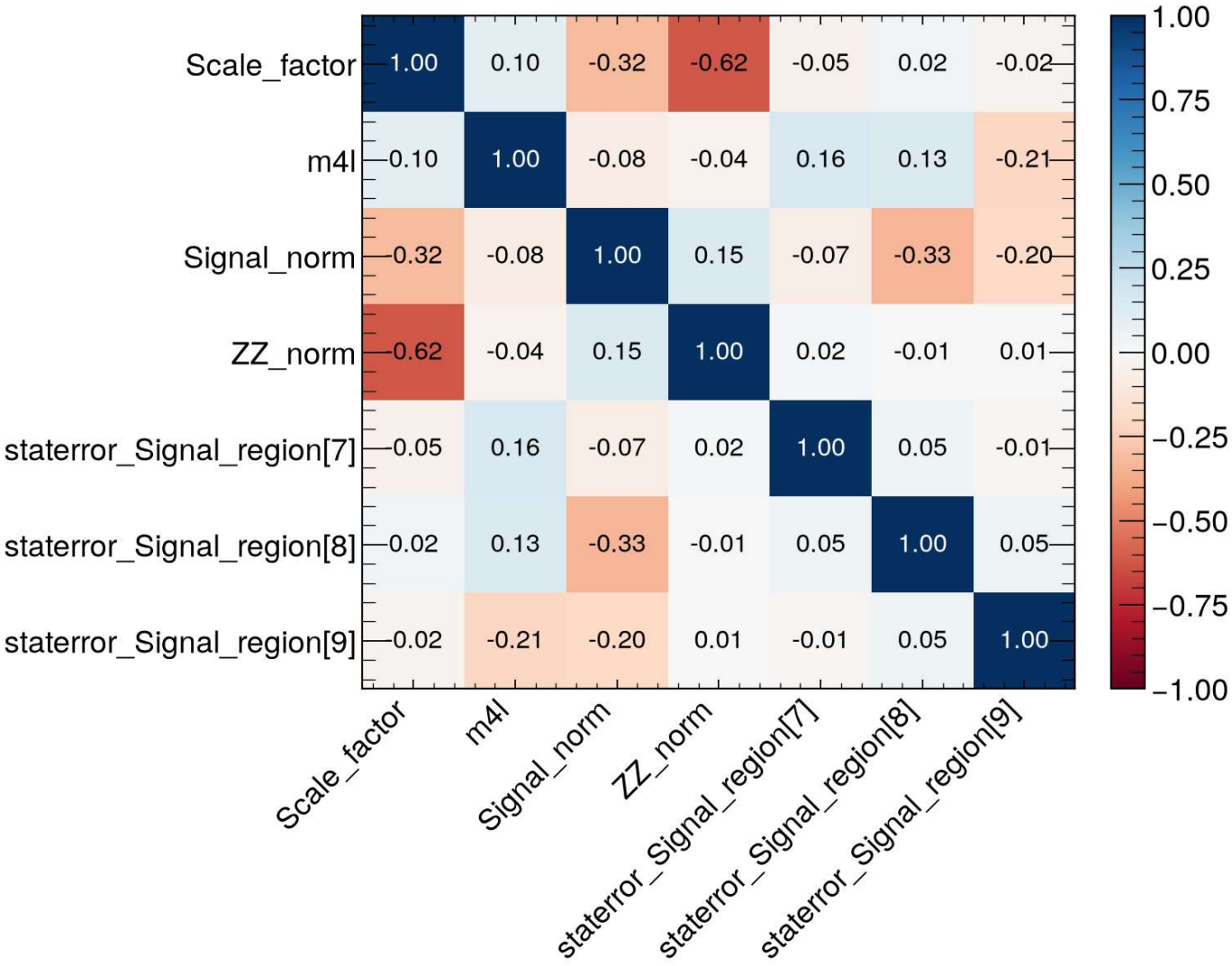
```
# %%
FIGURES_DIR.mkdir(parents=True, exist_ok=True)
mplhep.style.use("ATLAS")


def save_figure(figure, name: str) -> None:
    """Save a Matplotlib figure in both requested formats and close it."""
    figure.savefig(FIGURES_DIR / f"{name}.pdf", bbox_inches="tight")
    figure.savefig(FIGURES_DIR / f"{name}.png", dpi=160, bbox_inches="tight")
    plt.close(figure)


# %%
figure, axis = plt.subplots(figsize=(8, 6))
nominal_mc = signal_region["mc"][:, :, "nominal"]
nominal_mc.stack("process").plot(
    stack=True,
    histtype="fill",
    edgecolor="black",
    linewidth=0.7,
    color=["#7b2cbf", "#e63946", "#80b1d3"],
    ax=axis,
)
mplhep.histplot(
    signal_region["data"],
    histtype="errorbar",
    color="black",
    label=DATA,
    ax=axis,
)
axis.set_xlabel(r"Four-lepton invariant mass $m_{4\ell}$ [GeV]")
axis.set_ylabel("Events / 5 GeV")
axis.set_title("H -> ZZ* -> 4l signal region")
axis.legend(frameon=False)
save_figure(figure, "m4l")


# %%
figure, axis = plt.subplots(figsize=(8, 6))
for variation, label, color in (
    ("nominal", "Nominal", "black"),
    ("scale_factor_up", "Scale factor +1 sigma", "#d62728"),
    ("scale_factor_down", "Scale factor -1 sigma", "#1f77b4"),
):
    mplhep.histplot(
```

```
        signal_region["mc"][:, :, variation].project("m4l"),
        histtype="step",
        linewidth=2,
        label=label,
        color=color,
        ax=axis,
    )
axis.set_xlabel(r"Four-lepton invariant mass $m_{4\ell}$ [GeV]")
axis.set_ylabel("MC events / 5 GeV")
axis.set_title("Scale-factor systematic (all MC)")
axis.legend(frameon=False)
save_figure(figure, "scale-factor-variations")


# %%
figure, axis = plt.subplots(figsize=(8, 6))
for variation, label, color in (
    ("nominal", "Nominal", "black"),
    ("m4l_up", "m4l +1%", "#d62728"),
    ("m4l_down", "m4l -1%", "#1f77b4"),
):
    mplhep.histplot(
        signal_region["mc"][:, :, variation].project("m4l"),
        histtype="step",
        linewidth=2,
        label=label,
        color=color,
        ax=axis,
    )
axis.set_xlabel(r"Four-lepton invariant mass $m_{4\ell}$ [GeV]")
axis.set_ylabel("MC events / 5 GeV")
axis.set_title("Four-lepton mass systematic (all MC)")
axis.legend(frameon=False)
save_figure(figure, "m4l-variations")


# %%
figure, axis = plt.subplots(figsize=(8, 6))
for variation, label, color in (
    ("nominal", "Nominal", "black"),
    ("zz_norm_up", "ZZ normalization +10%", "#d62728"),
    ("zz_norm_down", "ZZ normalization -10%", "#1f77b4"),
):
    mplhep.histplot(
        signal_region["mc"][:, hist.loc(ZZSTAR), variation].project("m4l"),
        histtype="step",
```

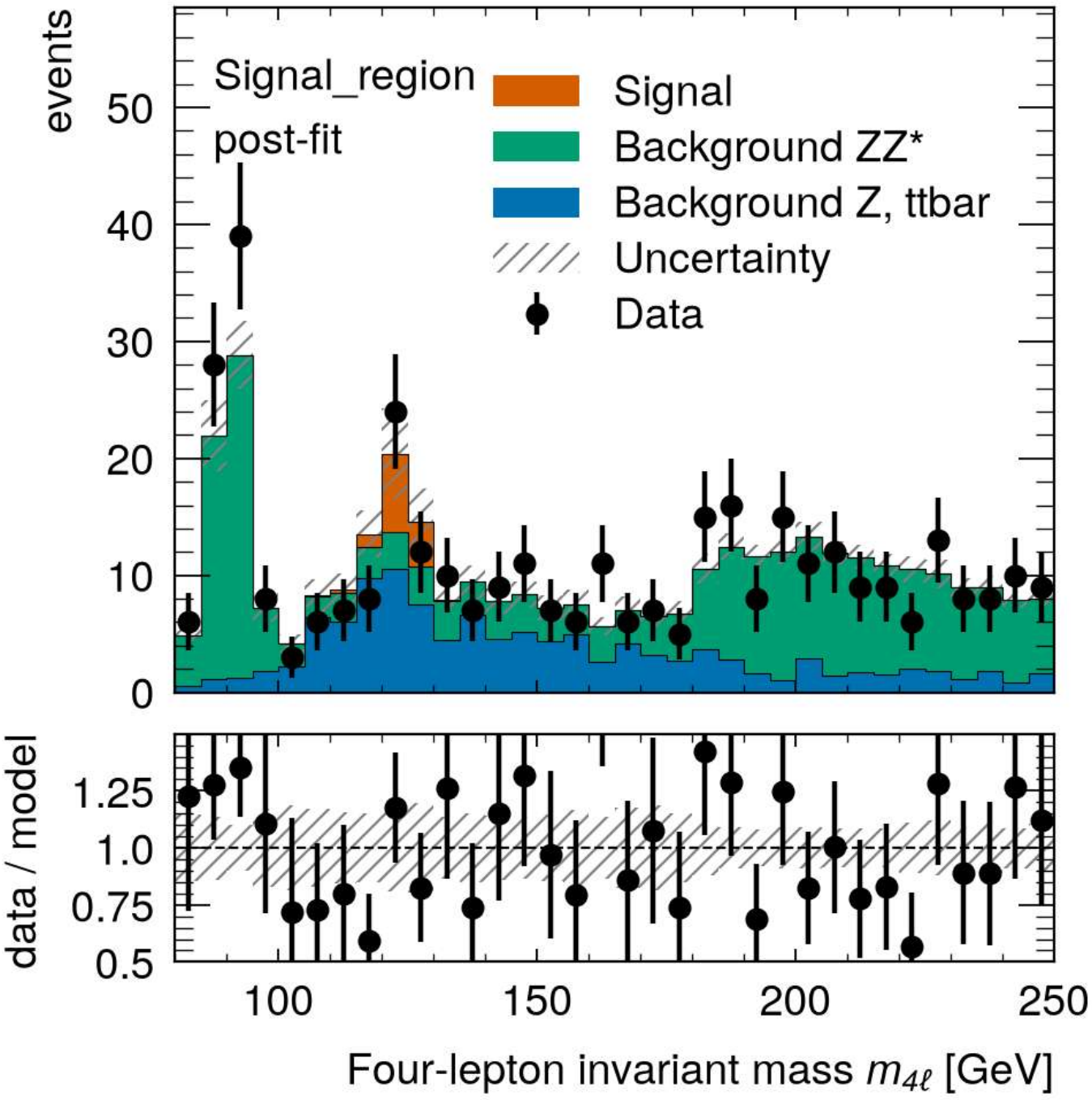
```
        linewidth=2,
        label=label,
        color=color,
        ax=axis,
    )
axis.set_xlabel(r"Four-lepton invariant mass $m_{4\ell}$ [GeV]")
axis.set_ylabel("ZZ* events / 5 GeV")
axis.set_title("ZZ* normalization systematic")
axis.legend(frameon=False)
save_figure(figure, "zz-normalization-variations")


# %%
ROOT_SAMPLE_NAMES = {Z_TTBAR: "Z_tt", ZZSTAR: "ZZ", SIGNAL: "signal"}
ROOT_VARIATION_SUFFIXES = {
    "scale_factor_up": "_SF_up",
    "scale_factor_down": "_SF_down",
    "m4l_up": "_m4l_up",
    "m4l_down": "_m4l_down",
}
with uproot.recreate(HISTOGRAM_FILE) as root_file:
    root_file["data"] = signal_region["data"]
    for process_name, root_name in ROOT_SAMPLE_NAMES.items():
        root_file[root_name] = signal_region["mc"][:, hist.loc(process_name), "nominal"]
        for variation, suffix in ROOT_VARIATION_SUFFIXES.items():
            root_file[root_name + suffix] = signal_region["mc"][
                :, hist.loc(process_name), variation
            ]
    root_file["ZZ_ZZ_norm_up"] = signal_region["mc"][:, hist.loc(ZZSTAR), "zz_norm_up"]
    root_file["ZZ_ZZ_norm_down"] = signal_region["mc"][
        :, hist.loc(ZZSTAR), "zz_norm_down"
    ]

print(f"Plotting completed; figures are in {FIGURES_DIR}")
print(f"Exported statistical templates to {HISTOGRAM_FILE}")

if STOP_AFTER == "plotting":
    raise SystemExit(0)


# %% [markdown]
# ## 4. Statistical analysis
#
# Collect and postprocess the ROOT templates, build the HistFactory workspace,
# fit the signal normalization POI, and save the fit diagnostics and result.
```

```
# %%
cabinetry_config = cabinetry.configuration.load(str(CONFIG_FILE))
print("Cabinetry samples:", cabinetry_config["Samples"])
print("Cabinetry systematics:", cabinetry_config["Systematics"])
cabinetry.templates.collect(cabinetry_config)
cabinetry.templates.postprocess(cabinetry_config)
workspace = cabinetry.workspace.build(cabinetry_config)
cabinetry.workspace.save(workspace, str(WORKSPACE_FILE))


# %%
model, data = cabinetry.model_utils.model_and_data(workspace)
fit_results = cabinetry.fit.fit(model, data)


# %%
cabinetry.visualize.pulls(
    fit_results,
    exclude="Signal_norm",
    close_figure=False,
    save_figure=False,
)
pull_figure = plt.gcf()
save_figure(pull_figure, "pull_plot")


# %%
cabinetry.visualize.correlation_matrix(
    fit_results,
    pruning_threshold=0.15,
    close_figure=False,
    save_figure=False,
)
matrix_figure = plt.gcf()
save_figure(matrix_figure, "matrix")


# %%
postfit_prediction = cabinetry.model_utils.prediction(model, fit_results=fit_results)
plot_config = {
    "Regions": [
        {
            "Name": "Signal_region",
            "Binning": list(np.linspace(BIN_EDGE_LOW, BIN_EDGE_HIGH, NUM_BINS + 1)),
        }
    ]
```

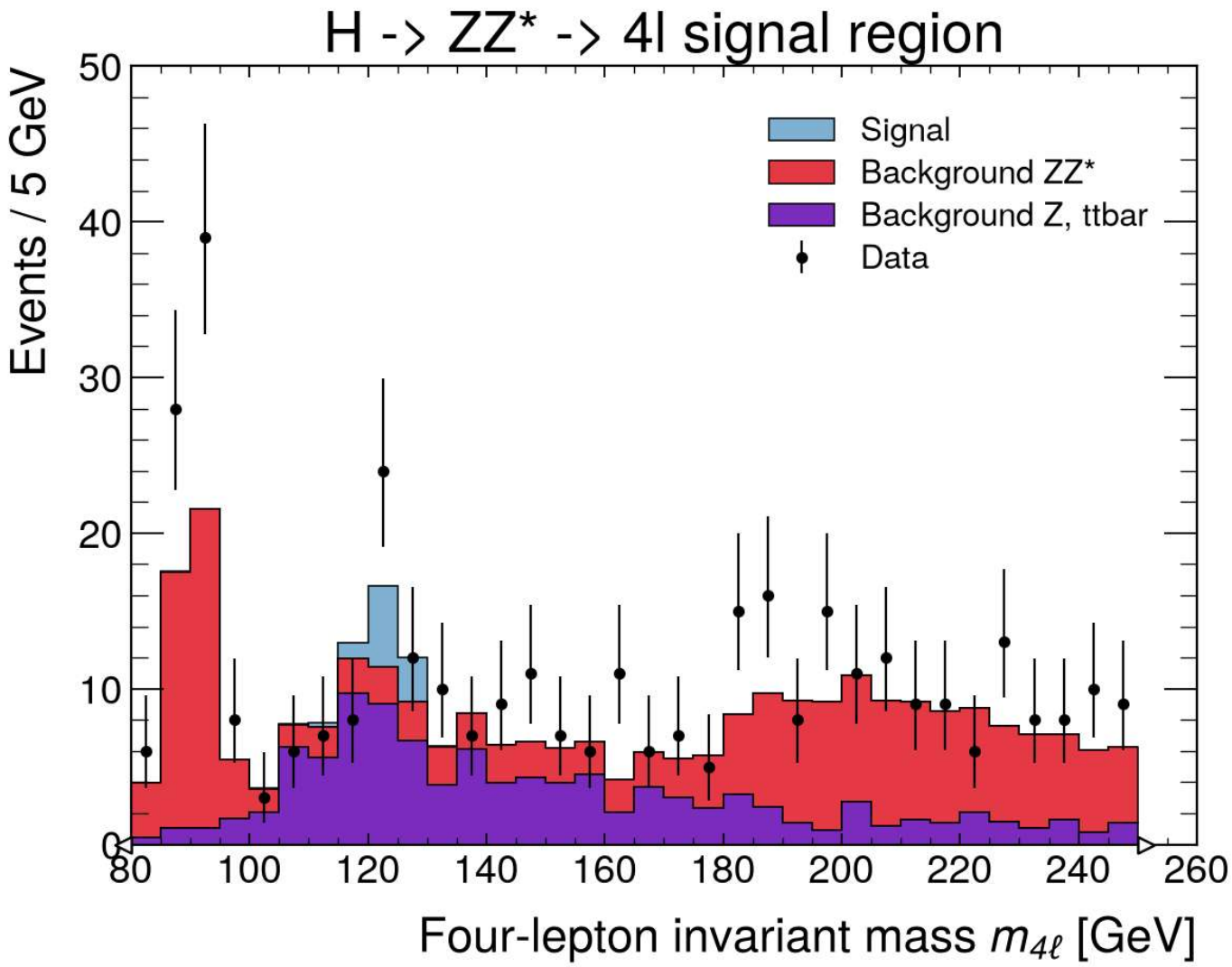
```
}
comparison_figures = cabinetry.visualize.data_mc(
    postfit_prediction,
    data,
    config=plot_config,
    close_figure=False,
    save_figure=False,
)
model_comparison_figure = comparison_figures[0]["figure"]
model_comparison_figure.axes[-1].set_xlabel(
    r"Four-lepton invariant mass $m_{4\ell}$ [GeV]"
)
save_figure(model_comparison_figure, "model-comp")


# %%
poi_index = fit_results.labels.index("Signal_norm")
poi_value = float(fit_results.bestfit[poi_index])
poi_error = float(fit_results.uncertainty[poi_index])
with RESULTS_FILE.open("w", encoding="utf-8") as results_stream:
    json.dump({"poi_val": poi_value, "poi_err": poi_error}, results_stream, indent=2)
    results_stream.write("\n")

print(
    f"Statistical analysis completed: Signal_norm = {poi_value:.4f} +/- {poi_error:.4f}"
)
print(f"Saved fit workspace to {WORKSPACE_FILE} and fit result to {RESULTS_FILE}")
```



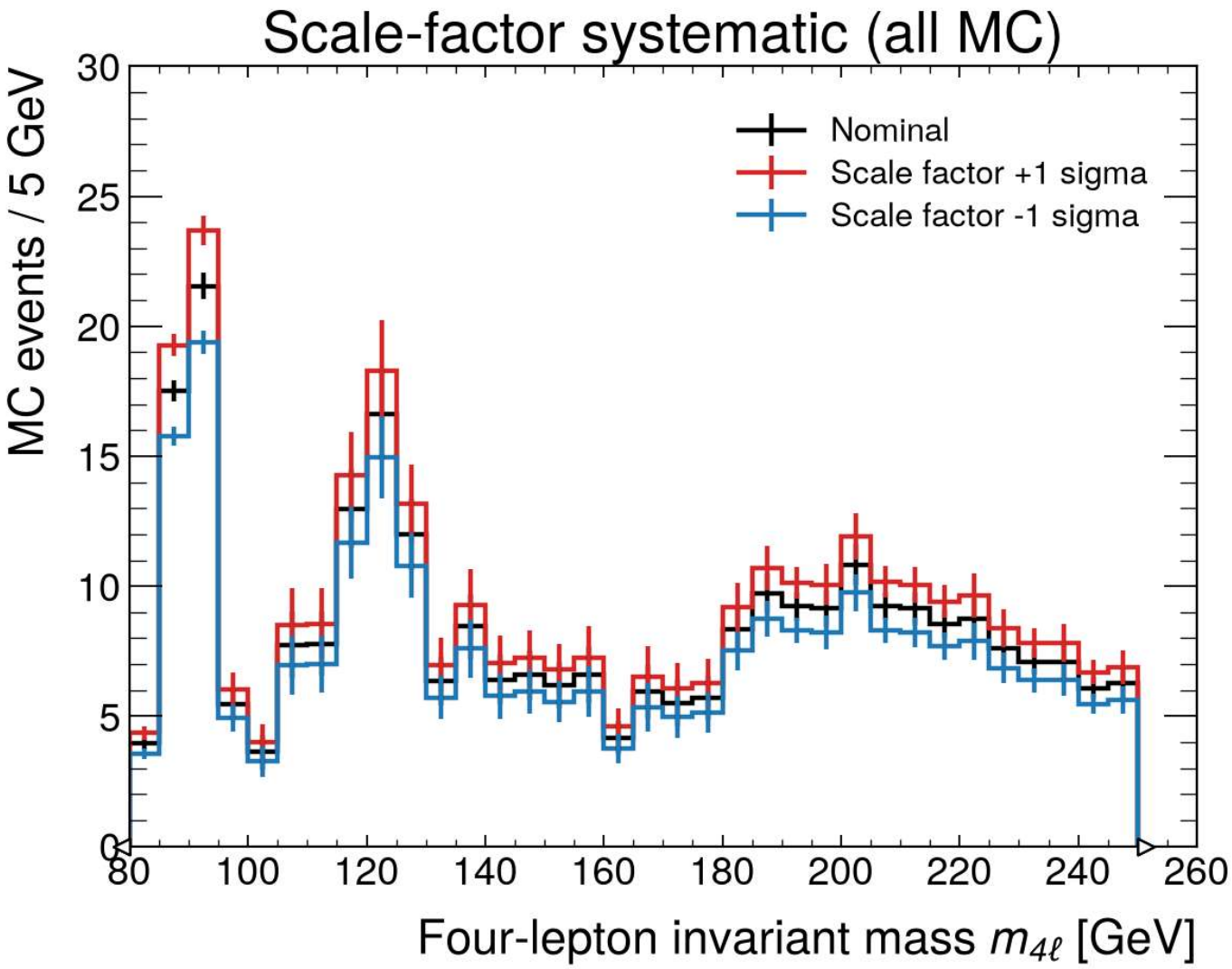
- Source: `figures/matrix.png`
- Size: 210,556 bytes



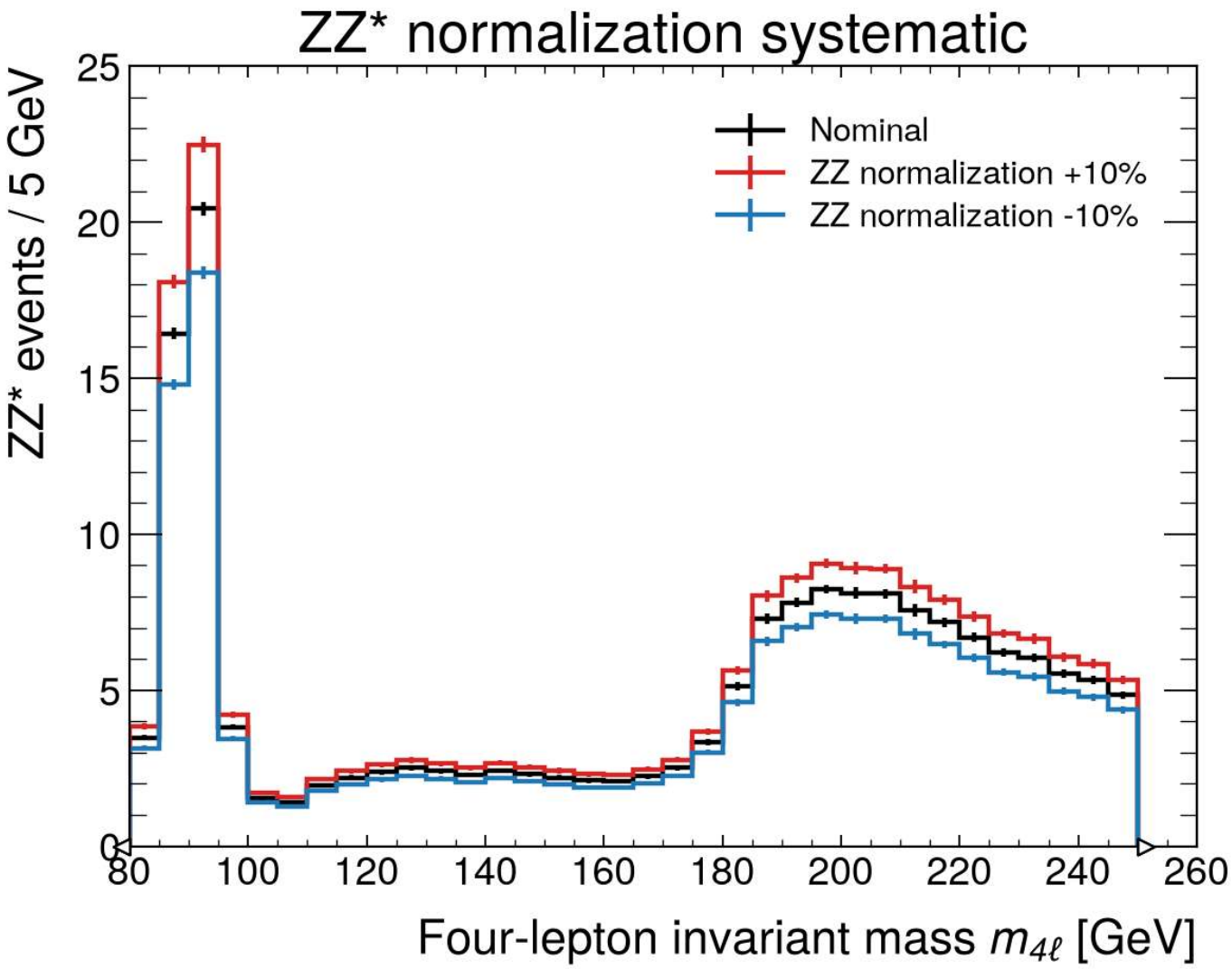
- Source: `figures/model-comp.png`
- Size: 128,826 bytes



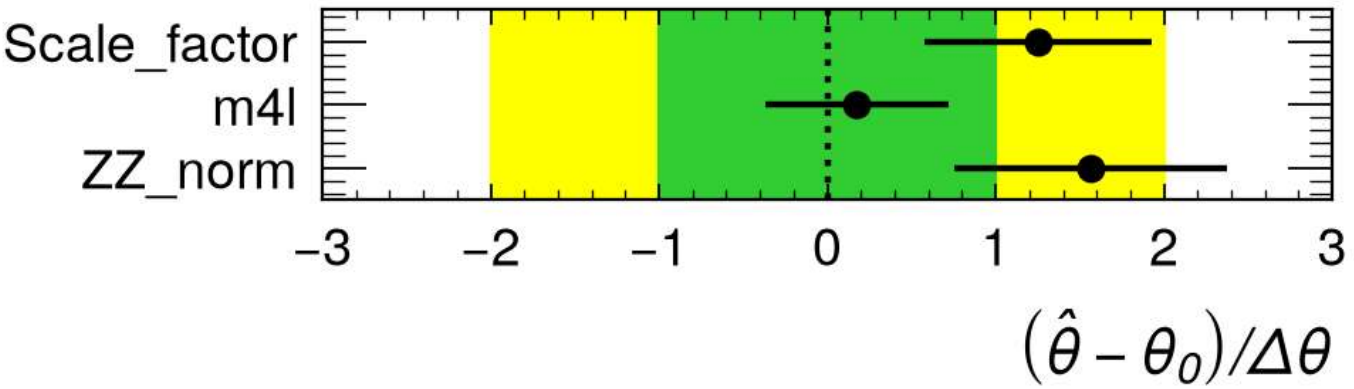
- Source: `figures/m4l.png`
- Size: 95,299 bytes



- Source: `figures/scale-factor-variations.png`
- Size: 107,668 bytes

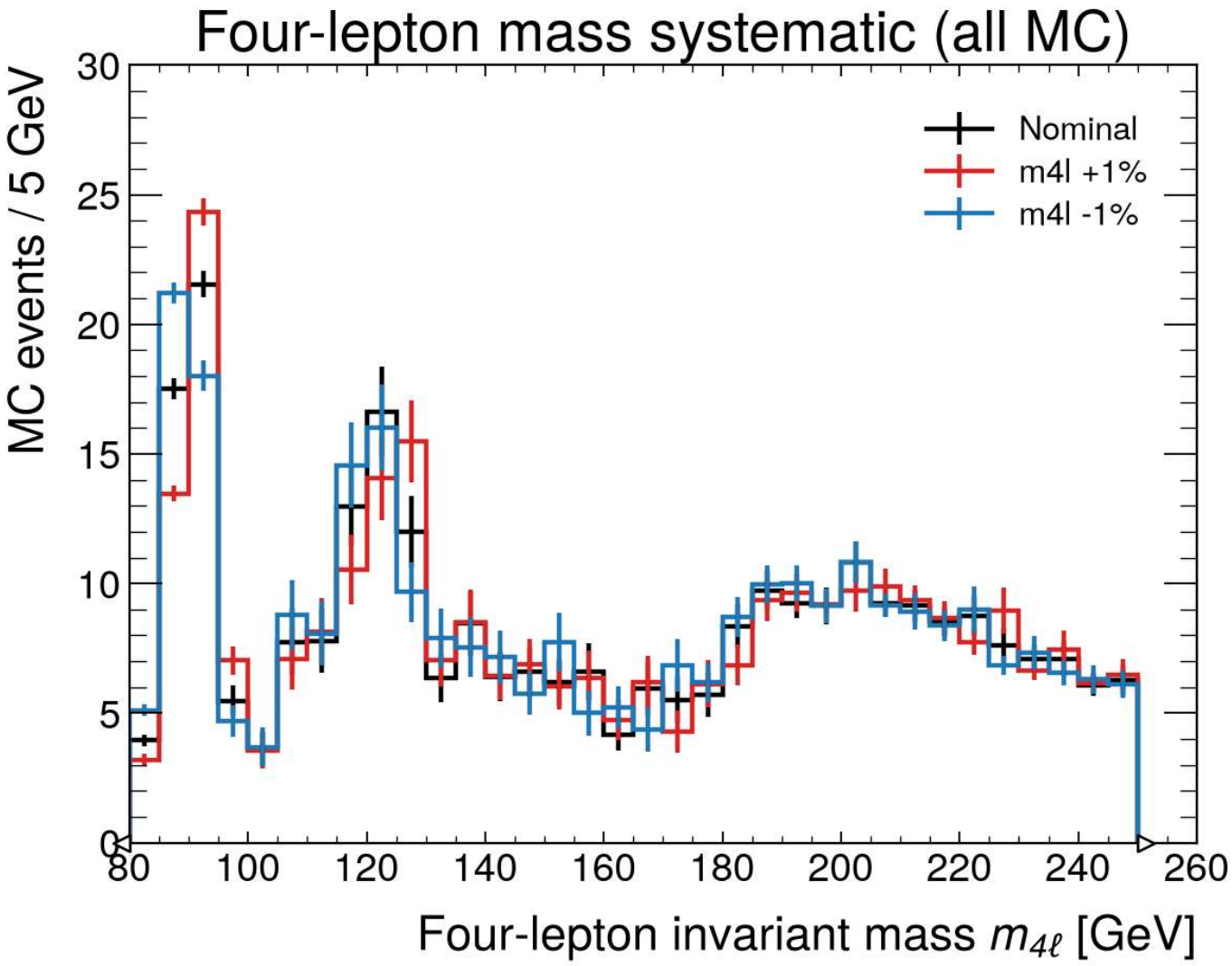


- Source: `figures/zz-normalization-variations.png`
- Size: 92,336 bytes



- Source: `figures/pull_plot.png`

- Size: 27,369 bytes



- Source: `figures/m4l-variations.png`
- Size: 98,472 bytes

▶ Complete Codex stderr

▶ Model activity

Schema version: `2`