Infrastructure

**. Core Components**

1. **Topology Loader**
   * Ingest your AdvancedGrid dataclasses (buses, lines, transformers, DERs, etc.) from JSON or Python modules.
2. **Power‑Flow Engine**
   * Use **Pandapower** to assemble a pp.Network and run runpp().
3. **Contingency Analyzer**
   * For each contingency (element outage),
     + Remove the element
     + Re‑run PF
     + Detect violations (line overloads, voltage limits)
4. **Mitigation Engine**
   * For each violation, apply a **rule‑based** action, for example:
     + **Re‑dispatch**: ramp up a nearby generator with spare capacity
     + **Load shed**: shed X MW at the most exposed bus/load class
     + **Re‑route**: change topology (close an alternate tie or FACTS setting)
   * Re‑run PF to confirm the violation is cleared.
5. **Result Collector & Excel Export**
   * Gather into a pandas DataFrame:  
     | contingency\_id | violated\_element | violation\_type | severity | mitigation\_type | target | amount\_mw | post\_pf\_status |
   * Write to an .xlsx via DataFrame.to\_excel()

**2. Directory Layout & Dependencies**

arduino

CopyEdit

grid-sim/

├── data/

│ └── topology.json # your blended-topology export

├── grid/

│ └── advanced\_grid.py # classes you posted

├── run\_sim.py # orchestrator main script

├── mitigation\_rules.py # defines rule‑based mitigation functions

├── requirements.txt # pandapower, pandas, openpyxl, concurrent‑futures

└── outputs/

└── contingency\_results.xlsx

**requirements.txt**

shell

CopyEdit

pandapower>=2.11

pandas

openpyxl

**3. Example Orchestrator (run\_sim.py)**

python

CopyEdit

import json

import pandas as pd

import pandapower as pp

from concurrent.futures import ThreadPoolExecutor

from grid.advanced\_grid import AdvancedGrid, Contingency

from mitigation\_rules import apply\_mitigation

# 1. Load topology

with open("data/topology.json") as f:

topo = json.load(f)

grid = AdvancedGrid.from\_json(topo)

# 2. Build base pandapower network

def build\_pp\_net(grid):

net = pp.create\_empty\_network()

# ... loop grid.buses, grid.lines, grid.transformers, grid.generators ...

return net

base\_net = build\_pp\_net(grid)

# 3. Analyze one contingency

def analyze\_cont(cont: Contingency):

net = base\_net.deepcopy()

cont.remove\_from(net) # stub: remove lines/buses

pp.runpp(net)

violations = []

# detect overloads

for idx, row in net.res\_line.iterrows():

if row.loading\_percent > 100:

violations.append(("line", idx, row.loading\_percent))

results = []

for vtype, eid, severity in violations:

action = apply\_mitigation(net, vtype, eid, severity)

# re‑run PF to confirm

pp.runpp(net)

status = "cleared" if max(net.res\_line.loading\_percent) <= 100 else "still\_violation"

results.append({

"contingency\_id": cont.id,

"violated\_element": eid,

"violation\_type": vtype,

"severity": severity,

"mitigation\_type": action["type"],

"target": action["target"],

"amount\_mw": action["amount"],

"post\_pf\_status": status

})

return results

# 4. Parallelize all contingencies

all\_conts = list(grid.contingencies.values())

with ThreadPoolExecutor(max\_workers=8) as ex:

futures = [ex.submit(analyze\_cont, c) for c in all\_conts]

all\_results = []

for f in futures:

all\_results.extend(f.result())

# 5. Export to Excel

df = pd.DataFrame(all\_results)

df.to\_excel("outputs/contingency\_results.xlsx", index=False)

print("Wrote", len(df), "rows to outputs/contingency\_results.xlsx")

**4. Mitigation Rules Example (mitigation\_rules.py)**

python

CopyEdit

def apply\_mitigation(net, vtype, eid, severity):

# Simple rule: if line overload, shed load at nearest bus by 10% of overload

if vtype == "line":

overload = severity - 100

# pick the “from\_bus” of that line

line = net.line.loc[eid]

target\_bus = line.from\_bus

shed\_amount = overload \* 1.1 # add 10% margin

# implement by reducing load in net.load

loads = net.load[net.load.bus == target\_bus]

if not loads.empty:

load\_idx = loads.index[0]

net.load.at[load\_idx, "p\_mw"] = max(0, net.load.at[load\_idx, "p\_mw"] - shed\_amount)

return {"type":"load\_shed", "target":target\_bus, "amount":shed\_amount}

# add more rules: generator redispatch, topology change...

return {"type":"none", "target":None, "amount":0}

**5. Next Steps**

* **Populate** AdvancedGrid.from\_json(...) so your Python class can rebuild the network.
* **Expand** apply\_mitigation with more granular actions (re‑dispatch, FACTS tuning, islanding).
* **Validate** end‑to‑end by inspecting outputs/contingency\_results.xlsx in Excel.

x\_ohm\_per\_km=line.x\_ohm\_per\_km,

c\_nf\_per\_km=0, # or line.capacitance

max\_i\_ka=line.rating\_normal/((net.sn\_mva/np.sqrt(3))\*10\*\*3),

name=line.id

)

Add transformers, loads, generators, external grid similarly.

Run power‑flow:

python

Copy

Edit

pp.runpp(net, algorithm="nr", init="flat", calculate\_voltage\_angles=True)

Error handling:

python

Copy

Edit

try:

pp.runpp(net)

except pp.LoadflowNotConverged:

logging.warning(f"PF failed for snapshot {snapshot\_id}")

return None

1.3 Contingency Analyzer

Purpose: For each outage scenario, remove elements, re‑solve PF, and detect violations.

Workflow:

python

Copy

Edit

def analyze\_contingency(base\_net, cont):

net = base\_net.deepcopy()

for elem in cont.elements:

if elem.startswith("L\_"):

net.line.drop(elem, inplace=True)

elif elem.startswith("G"):

net.gen.loc[elem, "in\_service"] = False

# handle transformers similarly

pp.runpp(net)

return detect\_violations(net)

Violation Detection:

python

Copy

Edit

def detect\_violations(net):

violations = []

# Thermal

for idx, row in net.res\_line.iterrows():

if row.loading\_percent > 100:

violations.append({

"violated\_element": idx,

"violation\_type": "thermal\_overload",

"severity": row.loading\_percent

})

# Voltage

for idx, row in net.res\_bus.iterrows():

if not 0.95 <= row.vm\_pu <= 1.05:

violations.append({

"violated\_element": idx,

"violation\_type": "voltage\_limit",

"severity": row.vm\_pu

})

return violations

Performance:

Use ThreadPoolExecutor or ProcessPoolExecutor to parallelize dozens or hundreds of scenarios.

Profile PF times and tune solver parameters (max\_iteration, tolerance).

1.4 Mitigation Engine

Purpose: Given a set of violations, generate and execute a rule‑based action plan to restore limits.

Action Plan Factory:

python

Copy

Edit

def plan\_actions(violations, grid, net):

plans = []

for v in violations:

if v["violation\_type"] == "thermal\_overload":

plans.append(plan\_line\_overload(v, grid, net))

elif v["violation\_type"] == "voltage\_limit":

plans.append(plan\_voltage\_fix(v, grid, net))

# …

return plans

Example Rule: Redispatch

python

Copy

Edit

def plan\_line\_overload(violation, grid, net):

line = grid.lines[violation["violated\_element"]]

# find nearest generator with spare capacity

candidates = sorted(net.gen.itertuples(),

key=lambda g: distance(grid.buses[g.bus], line))

for g in candidates:

spare = g.max\_p\_mw - g.p\_mw

if spare > 0:

return {

"type": "redispatch",

"target": g.Index,

"parameters": {"delta\_mw": min(spare, violation["severity"] \* 0.1)}

}

return {"type":"none"}

Execution Harness:

python

Copy

Edit

def execute\_plan(net, plan):

for step in plan:

if step["type"] == "redispatch":

net.gen.at[step["target"], "p\_mw"] += step["parameters"]["delta\_mw"]

elif step["type"] == "load\_shed":

# similar logic

pp.runpp(net) # confirm limits

return net

1.5 Result Collector & Excel Export

Purpose: Aggregate all scenario + violation + mitigation outcomes into a single table.

DataFrame Schema:

python

Copy

Edit

columns=[

"scenario\_id","element\_id","violation\_type","severity",

"mitigation\_type","target","amount\_mw","post\_pf\_status"

]

Collection Loop:

python

Copy

Edit

records = []

for cont in contingencies:

violations = analyze\_contingency(base\_net, cont)

if not violations:

records.append({"scenario\_id":cont.id, "violation\_type":"none"})

continue

plans = plan\_actions(violations, grid, base\_net)

for plan in plans:

net2 = execute\_plan(base\_net.deepcopy(), plan)

status = "cleared" if no\_violations(net2) else "residual"

records.append({

"scenario\_id":cont.id,

\*\*violation, # unpack element\_id, severity...

"mitigation\_type": plan["type"],

"target": plan.get("target"),

"amount\_mw": plan["parameters"].get("delta\_mw"),

"post\_pf\_status": status

})

Export:

python

Copy

Edit

df = pd.DataFrame(records)

df.to\_excel("outputs/contingency\_results.xlsx", index=False)

2. Directory Layout & Dependencies

bash

Copy

Edit

grid-sim/

├── data/

│ ├── topology.json # blended-topology export

│ └── contingencies/ # one JSON per scenario

├── grid/

│ └── advanced\_grid.py # dataclasses + from\_json()

├── simulation/

│ ├── build\_network.py # the Pandapower assembler

│ ├── contingency\_analyzer.py # analyze\_contingency, detect\_violations

│ └── mitigation\_engine.py # plan\_actions, execute\_plan

├── run\_sim.py # top-level orchestrator (Section 1–5)

├── mitigation\_rules.py # shared rule definitions

├── requirements.txt

└── outputs/

├── contingency\_results.xlsx

└── logs/

└── run\_YYYYMMDDTHHMMSS.log

requirements.txt

shell

Copy

Edit

pandapower>=2.11

pandas

openpyxl

numpy

3. Example Orchestrator (run\_sim.py)

Putting it all together:

python

Copy

Edit

import json, logging

from datetime import datetime

import pandas as pd

from concurrent.futures import ThreadPoolExecutor

from grid.advanced\_grid import AdvancedGrid

from simulation.build\_network import build\_network

from simulation.contingency\_analyzer import analyze\_contingency

from simulation.mitigation\_engine import plan\_actions, execute\_plan

# Logging setup

logging.basicConfig(level=logging.INFO)

# 1. Load topology & build base network

grid = AdvancedGrid.from\_json("data/topology.json")

base\_net = build\_network(grid)

# 2. Load scenarios

with open("data/scenarios\_index.json") as f:

scenarios = json.load(f) # list of {"id","type","elements",...}

# 3. Run analysis

records = []

def process\_scenario(sc):

recs = []

violations = analyze\_contingency(base\_net, sc)

plans = plan\_actions(violations, grid, base\_net)

for plan in plans:

net2 = execute\_plan(base\_net.deepcopy(), plan)

status = "cleared" if not detect\_violations(net2) else "residual"

recs.append({

"scenario\_id": sc["id"],

\*\*violations[0], # first violation details

"mitigation\_type": plan["type"],

"target": plan.get("target"),

"amount\_mw": plan["parameters"].get("delta\_mw"),

"post\_pf\_status": status

})

return recs

with ThreadPoolExecutor(max\_workers=8) as ex:

for result in ex.map(process\_scenario, scenarios):

records.extend(result)

# 4. Export results

df = pd.DataFrame(records)

df.to\_excel("outputs/contingency\_results.xlsx", index=False)

logging.info(f"Wrote {len(df)} records.")

4. Mitigation Rules Example (mitigation\_rules.py)

python

Copy

Edit

# Simple, extensible rule set

def shed\_load(net, bus, amount\_mw):

loads = net.load[net.load.bus == bus]

if not loads.empty:

idx = loads.index[0]

net.load.at[idx, "p\_mw"] = max(0, net.load.at[idx, "p\_mw"] - amount\_mw)

return {"type":"load\_shed", "target":bus, "parameters":{"delta\_mw":amount\_mw}}

def redispatch(net, gen, delta\_mw):

net.gen.at[gen, "p\_mw"] += delta\_mw

return {"type":"redispatch", "target":gen, "parameters":{"delta\_mw":delta\_mw}}

# You can import these in your mitigation\_engine to build plans

5. Next Steps

Populate all stubbed methods (AdvancedGrid.from\_json, build\_network, etc.).

Unit‑test each component in simulation/ with small sample topologies.

Profile PF and contingency loops—optimize solver settings or parallelism.

Iterate on your mitigation rules to match operator best practices and institutional guidelines.

With this expanded infrastructure, you’ll have a modular, scalable, and maintainable simulator—ready to generate the rich contingency outputs you need for both manual analysis and downstream ML experimentation.