**3. Define Contingency Scenarios**

To systematically stress‑test your simulator, you need a clear catalog of both N‑1 (single‐element) and selective N‑2 (double‐element) events, along with realistic likelihoods and impact metrics.

**3.1 Scenario Enumeration**

1. **N‑1 (Single‑Element) Events**
   * **Line Outages**
     + Every EHV/transmission line in the network
     + Includes both inter‑regional ties and intra‑region feeders
   * **Generator Trips**
     + All online generators (coal, gas, nuclear, hydro, wind, solar)
     + Treat renewable CF variations separately from forced outages
   * **Transformer Failures**
     + All step‑down and grid transformers at substations
     + Include bus‑section and inter‑tie transformers
2. **Selective N‑2 (Two‑Element) Events**
   * **Adjacent Element Pairs**
     + Two lines sharing the same bus or corridor
     + Two generators in the same plant
   * **High‑Risk Combinations**
     + Largest generator + critical tie‑line
     + Dual transformer outage on a single substation
   * **Geographically Correlated Failures**
     + Two elements within the same region (e.g. same weather cell)

**3.2 Assigning Probabilities & Severities**

| **Element Type** | **Base Failure Rate (F₀)** | **Severity Metric** |
| --- | --- | --- |
| **Line** | λ\_line = 0.005·exp(age/30) per yr | MW capacity lost; customers affected |
| **Generator** | λ\_gen = forced‑outage rate by type: • Thermal: 5–10% yr⁻¹ • Hydro: 2–5% yr⁻¹ • Renewables: intermittency profile | % of capacity; ramp recovery time |
| **Transformer** | λ\_xfmr = 0.01·age\_factor (yrs) | % of net transfer capability lost |

* **Normalized Probability**  
  For each element *i*:

Pi(outage in year)  =  λi∑jλj P\_i(\text{outage in year}) \;=\; \frac{\lambda\_i}{\sum\_j \lambda\_j}Pi​(outage in year)=∑j​λj​λi​​

* **Severity Distribution**  
  Draw severity *S* from a log‑normal or Weibull fit to historic outage MW losses  
  (e.g. mean = 50 MW, σ = 20 MW for lines; adjust per equipment class).

**3.3 Scenario Metadata Structure**

Store each scenario in a JSON schema like:

json

CopyEdit

{

"id": "N1\_LINE\_L\_1\_2",

"type": "N-1",

"elements": ["L\_1\_2"],

"probability": 0.00012,

"severity\_dist": {

"distribution": "lognormal",

"mean": 50,

"sigma": 0.3

}

}

For N‑2:

json

CopyEdit

{

"id": "N2\_GEN\_G1\_LINE\_L\_10\_11",

"type": "N-2",

"elements": ["G1","L\_10\_11"],

"probability": 0.0000015,

"severity\_dist": { … }

}

**3.4 Implementation Tips**

* **Automate Generation**
  + Loop over all single elements for N‑1
  + For N‑2, apply filters (same region, high‑impact) to limit combinatorial explosion
* **Weighting**
  + Use each element’s failure rate λ to compute joint probability for N‑2 as Pij=Pi×PjP\_{ij}=P\_i\times P\_jPij​=Pi​×Pj​ (assuming independence)
* **Severity Sampling**
  + When simulating, draw a sever­ity factor from the specified distribution to scale the violation thresholds (e.g. overload margin, MW shed).

**3.1 Scenario Enumeration (Exhaustive & Targeted)**

**A. N‑1 (Single‑Element) Events**

You’ll generate one scenario per element:

1. **Line Outages**
   * Loop over every transmission and feeder line in your AdvancedGrid.lines dictionary.
   * Tag each line as “tie” or “intra‑region” for downstream filtering.
2. **Generator Trips**
   * Include all online generators (gen.online == True).
   * For renewables (wind/solar), distinguish between:
     + **Forced trips** (treat as outage)
     + **Capacity fluctuations** (modeled by CF profiles; *not* here)
3. **Transformer Failures**
   * Every transformer in AdvancedGrid.transformers.
   * Include both step‑down (transmission→distribution) and inter‑tie transformers.

**Code Sketch: N‑1 Enumeration**

python

CopyEdit

def generate\_n1\_scenarios(grid):

scenarios = []

# Lines

for line in grid.lines.values():

scenarios.append({

"id": f"N1\_LINE\_{line.id}",

"type": "N-1",

"elements": [line.id]

})

# Generators

for gen in grid.generators.values():

if gen.online:

scenarios.append({

"id": f"N1\_GEN\_{gen.id}",

"type": "N-1",

"elements": [gen.id]

})

# Transformers

for xf in grid.transformers.values():

scenarios.append({

"id": f"N1\_XFMR\_{xf.id}",

"type": "N-1",

"elements": [xf.id]

})

return scenarios

**B. Selective N‑2 (Double‑Element) Events**

Brute‑forcing all pairs (O(n²)) is infeasible at ~2 000 elements. Instead apply **filters**:

1. **Adjacent Pairs**
   * Two lines sharing a common bus: find line\_a and line\_b where line\_a.to\_bus == line\_b.to\_bus or similar.
   * Two generators on the same bus or in the same plant region.
2. **High‑Risk Combinations**
   * **Top‑capacity generator + critical tie‑line**: sort generators by capacity\_mw and lines by transfer\_limit\_mw, pair the top 5 of each.
   * **Transformer duos**: any two transformers in the same substation.
3. **Geographically Correlated Failures**
   * Group elements by region (e.g. weather cell), then pick random or top‑k pairs within each group.

**Code Sketch: N‑2 Generation**

python

CopyEdit

from itertools import combinations

def generate\_n2\_scenarios(grid, max\_pairs=5000):

scenarios = []

# Adjacent line pairs

bus\_to\_lines = {}

for line in grid.lines.values():

bus\_to\_lines.setdefault(line.from\_bus, []).append(line.id)

bus\_to\_lines.setdefault(line.to\_bus, []).append(line.id)

for bus, lines in bus\_to\_lines.items():

for a,b in combinations(lines, 2):

scenarios.append({"id":f"N2\_LINES\_{a}\_{b}","type":"N-2","elements":[a,b]})

# Top capacity generator + top tie lines

top\_gens = sorted(grid.generators.values(), key=lambda g: g.capacity\_mw, reverse=True)[:5]

tie\_lines = [l for l in grid.lines.values() if l.transfer\_limit\_mw]

top\_lines = sorted(tie\_lines, key=lambda l: l.transfer\_limit\_mw, reverse=True)[:5]

for g in top\_gens:

for l in top\_lines:

scenarios.append({"id":f"N2\_GEN\_{g.id}\_LINE\_{l.id}","type":"N-2","elements":[g.id, l.id]})

# Limit total

return scenarios[:max\_pairs]

**3.2 Assigning Probabilities & Severity Distributions**

**A. Base Failure Rates (λ)**

| **Element** | **Failure Rate λ (per year)** |
| --- | --- |
| **Line** | 0.005 × exp(age\_years / 30) |
| **Generator** | • Thermal: 0.05–0.10 • Hydro: 0.02–0.05 • Renewables: 0.10 (intermittency) |
| **Transformer** | 0.01 × (age\_years / 10) |

python

CopyEdit

def line\_failure\_rate(line):

return 0.005 \* math.exp(line.age\_years / 30)

def gen\_failure\_rate(gen):

base = {"coal":0.05, "gas":0.07, "nuclear":0.03, "hydro":0.03, "wind":0.10, "solar":0.10}

return base.get(gen.type, 0.05)

def xfmr\_failure\_rate(xf):

return 0.01 \* (xf.age\_years / 10)

**B. Normalized Scenario Probability**

For each scenario “s” that affects elements {i}\{i\}{i}:

Ps=∏i∈sλi∑s′∏j∈s′λjP\_s = \frac{\prod\_{i\in s} \lambda\_i}{\sum\_{s'} \prod\_{j\in s'}\lambda\_j}Ps​=∑s′​∏j∈s′​λj​∏i∈s​λi​​

(so that ∑sPs=1\sum\_s P\_s = 1∑s​Ps​=1)

python

CopyEdit

lambdas = [line\_failure\_rate(l) for l in grid.lines.values()]

total = sum(lambdas)

for sc in n1\_scenarios:

elem = sc["elements"][0]

lam = line\_failure\_rate(grid.lines[elem])

sc["probability"] = lam / total

**C. Severity Distribution**

Model the **impact size** (e.g. MW lost) via a log‑normal or Weibull fit:

jsonc

CopyEdit

"severity\_dist": {

"distribution": "lognormal",

"mean\_mw": 50,

"sigma\_mw": 20

}

At run‑time, draw:

python

CopyEdit

import numpy as np

sev = np.random.lognormal(np.log(mean\_mw), sigma\_mw)

**3.3 Scenario Metadata Structure**

Define a **JSON schema** so all scenarios are validated and discoverable:

json

CopyEdit

{

"$schema":"http://json-schema.org/draft-07/schema#",

"title":"ContingencyScenario",

"type":"object",

"properties":{

"id": {"type":"string"},

"type": {"enum":["N-1","N-2"]},

"elements": {"type":"array","items":{"type":"string"}},

"probability": {"type":"number","minimum":0,"maximum":1},

"severity\_dist":{

"type":"object",

"properties":{

"distribution":{"enum":["lognormal","weibull"]},

"mean\_mw": {"type":"number"},

"sigma\_mw": {"type":"number"},

"shape": {"type":"number"},

"scale": {"type":"number"}

},

"required":["distribution"]

}

},

"required":["id","type","elements","probability","severity\_dist"]

}

Store each scenario in its own file under data/contingencies/, e.g.:

bash

CopyEdit

data/contingencies/N1\_LINE\_L\_1\_2.json

data/contingencies/N2\_GEN\_G1\_LINE\_L\_10\_11.json

**3.4 Implementation Tips & Best Practices**

1. **Automate & Version**
   * Write a generation script (generate\_scenarios.py) that outputs JSON and commits to Git.
   * Include a **topology hash** in each scenario for traceability.
2. **Filter & Prioritize**
   * Allow command‑line flags to restrict to N‑1, only line outages, or only high‑probability events.
   * Sort by probability × expected\_severity to pick the “top K” most critical scenarios.
3. **Severity Sampling**
   * During simulation, sample one or more draws from the severity\_dist and **scale** your violation thresholds accordingly (e.g. increase line loading limit by sampled MW).
4. **Independence & Correlation**
   * N‑2 probabilities assume independence: Pij=PiPjP\_{ij} = P\_i P\_jPij​=Pi​Pj​.
   * For geographically correlated failures, **boost** joint probabilities by a factor (e.g. × 1.5) to reflect common‑cause.
5. **Metadata Queries**
   * Build utility functions to filter scenarios by element type, region, or min/max probability.
   * E.g. get\_scenarios(filter={"type":"N-1", "elements\_contains":"L\_"}).