Conformalized Counterfactual LLM ×NS-3 ? End-to-End Run Guide
Last updated: 2025-09-16 14:36

Overview

This project connects an NS-3 LTE wireless scenario with an Ollama-served LLM and a conformal counterfactual pipeline. You can: (1) generate calibration/test data; (2) calibrate a distance threshold ??; (3) test by sampling counterfactual candidates until they fall within ??; and (4) visualize results.

Prerequisites

? macOS/Linux
? Python 3.10+ (recommended: a virtualenv)
? NS-3 (tested with 3.39): your tree should provide an ?ns3? runner (or ?waf? on older trees)
? Ollama running locally with a pulled model (e.g., llama3:latest, or a smaller/faster one)
? Build tools for NS-3 (cmake, clang/gcc) already configured

1) Clone / unpack the repo

```
$ unzip cf-llm-ns3-conformal.zip
$ cd cf-llm-ns3-conformal
```

2) Python environment and dependencies

```
$ python3 -m venv .venv
$ source .venv/bin/activate
$ pip install -r requirements.txt
```

3) Configure environment

```
$ cp .env.example .env
```

Edit .env:

```
  NS3_ROOT=/path/to/your/ns-3.39     (e.g., /Users/you/ns-3.39 or /Users/you/Desktop/NS/ns-3-dev)
  OLLAMA_HOST=http://localhost:11434
  OLLAMA_MODEL=llama3:latest         (or a faster one, e.g., llama3.1:8b-instruct)
  OUTPUT_DIR=outputs
  ALPHA=0.1
  METRIC=rouge                       (options: rouge, numeric)
  MAX_SAMPLES=50
```

Tip: Shell scripts do not read .env automatically. Either export in your shell:

```
$ export NS3_ROOT=/path/to/ns-3.39
```

or source the .env:

```
$ set -a; source .env; set +a
```

4) Build the NS-3 scenario

```
$ chmod +x scripts/patch_build_ns3.sh
$ ./scripts/patch_build_ns3.sh
```

This copies ns3/ran-sim.cc into $NS3_ROOT/scratch and builds it.

Sanity check:

```
$ $NS3_ROOT/ns3 run ran-sim --no-build -- --numUes=5 --scheduler=rr --trafficMbps=1.0 --duration=2
--rngRun=1 --output=/tmp/metrics.json
$ cat /tmp/metrics.json
```

5) (Optional) Warm up Ollama

```
$ ollama pull llama3:latest    # or your chosen model
$ ollama run llama3:latest "ok"
```

6) Generate calibration + test datasets

This runs the full LLM?NS-3?report pipeline for pairs (X, X?) with shared exogenous noise.
Outputs go to outputs/data/.
Example (smaller pilot first):

```
$ python -m cfllm.data_gen_calib --n-calib 10 --n-test 5 --alpha 0.1 --metric rouge --seed 123
```

Artifacts:
? outputs/data/calib.jsonl

? outputs/data/test.jsonl
? outputs/data/meta.json

Inspect prompts (no jq):

```
$ python - <<’PY’
import json, itertools
for i, line in enumerate(itertools.islice(open("outputs/data/calib.jsonl"), 5), 1):
    r = json.loads(line)
    print(f"{i}. X: {r[’X’]}\n   X’: {r[’X_prime’]}\n")
PY
```

7) Calibrate the threshold ??
Split-conformal threshold based on nonconformity s_i = d(Y_i, Y?_i,true),
with k = ceil((n+1)(1-?)), ?? = s_(k).

```
$ python -m cfllm.calibrate --alpha 0.1 --metric rouge
```

Output: outputs/calibration/calibration.json

Switch metric to numeric (no regeneration needed):

```
$ python -m cfllm.calibrate --metric numeric
```

8) Test on held-out pairs
Given (X, Y) and alternative X?, sample candidates under X? until d(Y, Y?^) ? ?? or
the budget is exhausted. Results go to outputs/test/.

```
$ python -m cfllm.test_cf --metric rouge --max-samples 8 --seed 123
```

Outputs:
? outputs/test/results.csv
? outputs/test/summary.json (includes tau, coverage, accept rate, avg samples, etc.)

9) Plotting (install matplotlib once)

```
$ pip install matplotlib
```

Quick plots:

```
$ python - <<’PY’
import json, pandas as pd, matplotlib.pyplot as plt
df = pd.read_csv("outputs/test/results.csv")
with open("outputs/test/summary.json") as f: summ = json.load(f)
tau = summ["tau"]
plt.figure(); plt.bar(df["idx"], df["samples_used"]); plt.xlabel("Test case idx"); plt.ylabel("Samples used")
plt.title("Samples used per test case"); plt.tight_layout();
plt.savefig("outputs/test/samples_per_case.png"); plt.close()
plt.figure(); plt.bar(["covered_truth","accepted"], [df["covered_truth"].sum(), df["accepted"].sum()])
plt.ylabel("Count"); plt.title("Coverage and acceptance counts"); plt.tight_layout()
plt.savefig("outputs/test/coverage_acceptance.png"); plt.close()
plt.figure(); df["dist_Y_Yp_true"].hist(bins=10); plt.axvline(tau, linestyle="--")
plt.xlabel("d(Y, Y’_true)"); plt.ylabel("Count"); plt.title("Truth distance distribution with tau")
plt.tight_layout(); plt.savefig("outputs/test/truth_distance_hist.png"); plt.close()
vals = pd.to_numeric(df["dist_Yp_accepted_to_true"], errors="coerce").dropna()
if len(vals): plt.figure(); vals.hist(bins=min(8, max(3, len(vals)))); plt.xlabel("d(Y’^acc, Y’_true)")
plt.ylabel("Count"); plt.title("Error of accepted vs truth"); plt.tight_layout()
plt.savefig("outputs/test/accepted_error_hist.png"); plt.close()
print("Saved plots under outputs/test/")
PY
```

10) Speed tips (recommended when scaling to ~100 samples)
? env_bridge: ensure ns3 runs with --no-build; add subprocess timeout (60s).
? Reuse the action for X? during sampling; vary only NS-3 rngRun. (Cheap & effective.)
? Cap candidate duration at test-time (e.g., 2?3s) to keep each attempt fast.
? Parallelize across test rows (ProcessPoolExecutor; e.g., --workers 4).
? Cache action_from_prompt(prompt) results on disk to avoid repeated LLM calls.
? Use a smaller Ollama model for faster generations.
? Use metric=numeric for faster comparisons and more meaningful numerical alignment.

11) Switching metrics
? ROUGE (default): distance = 1 ? ROUGE-L F1 (lower is better).
? numeric: extract numbers from Y and Y?, align by order, return average absolute diff.
To switch:
$ python -m cfllm.calibrate --metric numeric
$ python -m cfllm.test_cf --metric numeric --max-samples 8 --seed 123

12) Interpreting test summary
Example:
{"n": 5, "tau": 0.6818, "metric": "rouge", "coverage_truth": 0.6,
"accept_rate": 0.6, "avg_samples": 3.4, "avg_error_if_accepted": 0.6321}
? coverage_truth: fraction with d(Y, Y?_true) ? ?? (target ? 1?? in large samples).
? accept_rate: fraction where at least one candidate Y?^ met d ? ?? within budget.
? avg_samples: attempts per case (lower is faster).
? avg_error_if_accepted: for accepted cases, distance between Y?^ (accepted) and Y?_true.

13) Troubleshooting
? ?NS3_ROOT is not set?: export NS3_ROOT or source your .env with set -a.
? Build errors on ran-sim.cc: re-run scripts/patch_build_ns3.sh; ensure LTE/EPC/FlowMonitor are enabled.
? Slow/stuck runs: add --no-build; cap duration; add timeouts; warm up Ollama and a short NS-3 run.
? LLM JSON parse errors: cfllm/llm.py re-prompts once; switch to a more instruction-following model if needed.
? No accepts: increase ? (larger ??), use numeric metric, gentler edits, or increase sample budget.
? Plots need matplotlib: pip install matplotlib.

14) Command quick-reference
```
# Build once
export NS3_ROOT=/path/to/ns-3.39
./scripts/patch_build_ns3.sh

# Generate data (pilot)
python -m cfllm.data_gen_calib --n-calib 10 --n-test 5 --alpha 0.1 --metric rouge --seed 123

# Calibrate
python -m cfllm.calibrate --alpha 0.1 --metric rouge     # or --metric numeric

# Test
python -m cfllm.test_cf --metric rouge --max-samples 8 --seed 123

# Switch to numeric
python -m cfllm.calibrate --metric numeric
python -m cfllm.test_cf --metric numeric --max-samples 8 --seed 123
```

Notes
? All paths are relative to the repo root unless stated.
? You can freely regenerate data and recalibrate with different metrics/?.
? The conformal math is standard split-conformal using exchangeable (X, X?) pairs with shared noise to obtain Y?_true.