**Step 8 — Quantile-Buffer Robust Routing + Stochastic Evaluation**

**Abstract**

We introduced a fast robustification of the VRPTW by inflating each travel time with a fixed quantile buffer (×1.2). We then evaluated both the deterministic plans and the buffered plans under 100 stochastic traffic scenarios (global and link-level variability). On the subset successfully evaluated so far, the quantile method (Q120) achieves ≈99.84% on-time service with near-zero average tardiness, dramatically outperforming the deterministic plans (DET). Coverage of Q120 is not yet complete (only 3/56 instances produced valid evaluation rows), so the next action is to finish running/refreshing Q120 solutions for all instances and re-evaluate.

**What we did (method)**

1. **Quantile-buffer plans (Q120).**  
   We created “robust” time matrices by multiplying baseline arc times by **1.2** (a quick proxy for ~90th–95th percentile travel time), then solved with the same OR-Tools pipeline as Step 5/6 (time windows, capacity, service time), using:
   * --meta GLS, --time\_limit 30, --vehicle\_cost 10000.
   * Command (PowerShell):
   * python scripts\vrptw\_quantile.py --all --mult 1.2 --time\_limit 30 --vehicle\_cost 10000 --meta GLS
   * Output: JSON solutions per instance in data/solutions\_quantile/m1.2\_a0/ and a summary.csv there.
2. **Scenario-based evaluation (same test for all methods).**  
   We evaluated plans under **100 scenarios** with **common random numbers**:
   * **Global multiplicative factor** (per scenario) with CV=0.20 (correlated congestion).
   * **Link-level factor** (per arc, per scenario) with CV=0.10 (idiosyncratic noise).
   * Metrics per instance (computed across the 100 scenarios):
     + ontime\_mean (% of customers served within windows),
     + ontime\_p05, ontime\_p50, ontime\_p95,
     + tard\_mean (mean lateness, time-unit).
   * Command:
   * python scripts\evaluate\_plans.py `
   * --dirs "data\solutions\_ortools" "data\solutions\_quantile\m1.2\_a0" `
   * --labels DET Q120 `
   * --K 100 --seed 42 --cv\_global 0.20 --cv\_link 0.10

**Why this design?**

* The **quantile buffer** is the quickest “vite” robust method: one knob (multiplier) that usually boosts on-time performance with limited cost impact.
* **Stochastic evaluation** with common scenarios is the fairest way to compare methods; it tells us whether a plan actually stays on time under realistic variability rather than only in the deterministic world.

**What we created (artifacts)**

* **Solutions (Q120):** data/solutions\_quantile/m1.2\_a0/  
  (JSON per instance + summary.csv).
* **Evaluation outputs:**
  + data/reports/step8\_eval.csv — per-instance metrics for each method label (DET, Q120).
  + data/reports/step8\_eval\_by\_method.csv — method-level averages.

**Results (what we got)**

From your run (loaded from the two CSVs above):

**Method-level averages** (data/reports/step8\_eval\_by\_method.csv)

* **Q120:** ontime\_mean ≈ 99.8367%, tard\_mean ≈ 0.0132
* **DET:** ontime\_mean ≈ 42.7486%, tard\_mean ≈ 257.7964

**Coverage status (how many instances actually evaluated):**

* DET: **50** instances with valid metrics
* Q120: **3** instances with valid metrics  
  *(Many Q120 rows are present but have NaNs, which means those JSON plans didn’t evaluate correctly—likely missing/empty routes for some instances.)*

**Interpretation**

* Where Q120 produced valid solutions, **on-time % is essentially 100%** and **tardiness is ~0**, which is exactly what we expect from a 1.2 buffer.
* The **deterministic** plans collapse under variability (≈43% on-time on average, with large tardiness), confirming the need for robustification.
* The **next step** is to complete Q120 coverage (ensure a JSON solution for each of the 56 instances), then re-run the evaluation to get full family-level statistics (C/R/RC; 1 vs 2 horizon).

**Reproduction (exact commands)**

1. **Build buffered plans (Q120):**
2. python scripts\vrptw\_quantile.py --all --mult 1.2 --time\_limit 30 --vehicle\_cost 10000 --meta GLS
3. **Evaluate DET vs Q120 under the same scenarios:**
4. python scripts\evaluate\_plans.py --dirs "data\solutions\_ortools" "data\solutions\_quantile\m1.2\_a0" `
5. --labels DET Q120 --K 100 --seed 42 --cv\_global 0.20 --cv\_link 0.10
6. **Outputs to cite:**
   * data/reports/step8\_eval.csv
   * data/reports/step8\_eval\_by\_method.csv

**Quality checks & how to finish Step 8**

* **Check that you really have 56 Q120 JSONs:**
  + Folder should be data\solutions\_quantile\m1.2\_a0 with **56** \*.json.
  + If fewer: re-run vrptw\_quantile.py (you can increase --time\_limit to 60 and/or try --meta TABU) until all 56 are produced.
* **Re-evaluate** with the same command above to refresh the CSVs.
* (Optional) Try **other multipliers** (1.1 and 1.3) to see the cost vs on-time trade-off:
* python scripts\vrptw\_quantile.py --all --mult 1.1 --time\_limit 30 --vehicle\_cost 10000 --meta GLS
* python scripts\vrptw\_quantile.py --all --mult 1.3 --time\_limit 30 --vehicle\_cost 10000 --meta GLS
* python scripts\evaluate\_plans.py --dirs "data\solutions\_ortools" "data\solutions\_quantile\m1.1\_a0" "data\solutions\_quantile\m1.2\_a0" "data\solutions\_quantile\m1.3\_a0" --labels DET Q110 Q120 Q130 --K 100 --seed 42 --cv\_global 0.20 --cv\_link 0.10

**Takeaways (Step 8)**

* The **quantile-buffer (×1.2)** is a **simple, effective, and fast** robust method.
* **Preliminary** results show **huge on-time gains** (≈100% vs ≈43%) and **near-zero tardiness** for Q120—on the instances that evaluated correctly.
* Complete coverage will let us report family-level aggregates and choose the best multiplier for the full 56-instance set.

**What’s next**

* **Finish Q120 coverage** (56/56 plans) and **re-evaluate**.
* **Pick the best multiplier** (1.1/1.2/1.3) based on full results.
* Move to **Step 9 (SAA)** to compare a true stochastic optimizer against the quantile buffer.