**Executive summary (Step 9)**

We compared three planning approaches on the original 56 Solomon VRPTW instances under realistic travel-time variability: a deterministic baseline (DET), a fast **quantile buffer** plan (Q-buffer, 1.2×), and a **stochastic** Sample Average Approximation (SAA) planner at three conservatism levels. Plans were evaluated with the same 200 stochastic scenarios (CV\_global=0.20, CV\_link=0.10).  
**Deterministic plans collapsed under variability (≈40% on-time on average).**  
**Q-buffer lifted on-time to ≈99–100%** with a moderate cost increase.  
**SAA (K=16, β=0.3) dominated the trade-off**: it reached ~99.7% on-time **with the lowest cost among “champion” plans**, outperforming Q-buffer while keeping fleet size close to baseline. More conservative SAA settings (K=32, β=0.5 and K=64, β=0.7) added cost without meaningful on-time gains once we cross the ~99% service level.  
**Recommendation:** if compute is tight, use Q-buffer (m=1.2). For best cost at ≥99% on-time, use SAA with **K=16, β=0.3**.

**Step 9 — Stochastic planning (SAA) vs. deterministic and Q-buffer**

**1) Objective and motivation**

Until Step 8, plans were built with deterministic travel times (DET) or with a static risk buffer (Q-buffer). Real operations face correlated uncertainty; the questions for Step 9 were:

* *Can we optimize* ***with*** *the uncertainty rather than against it?*
* *What do we gain (on-time %) and what do we pay (distance/fleet)?*
* *How sensitive are results to the number of scenarios (K) and the risk-aversion weight (β)?*

**2) Methods and configurations**

* **DET** (deterministic OR-Tools): routes built on baseline travel times.
* **Q-buffer (Q120)**: solve on an inflated time matrix (mult=1.2), using the same OR-Tools pipeline. This is the fast “robustification” baseline from Step 8.
* **SAA (stochastic)**: we sample **K** travel-time scenarios per instance and evaluate candidate moves by **expected cost + β × expected tardiness** across those scenarios. We tested:
  + **SAA16-b0p3**: K=16, β=0.3 (least conservative, “value” setting)
  + **SAA32-b0p5**: K=32, β=0.5 (balanced)
  + **SAA64-b0p7**: K=64, β=0.7 (most conservative)
* **Common modeling** (all methods): time windows, service times, capacities, and a strong vehicle penalty (vehicle\_cost=10000) to discourage adding trucks unnecessarily. Metaheuristics used: GLS / TABU with time limits (30–60s per instance in your runs).

**3) Evaluation protocol (fair and comparable)**

* **Data:** the original **56 CSVs** (C/R/RC × {1,2}) used as-is.
* **Uncertainty model for evaluation:** **200 scenarios** per instance, *common* to all methods (same random seed), with correlated multiplicative noise:
  + **Global factor** per scenario (CV\_global=0.20) captures across-network conditions (peak/off-peak).
  + **Link factors** (CV\_link=0.10) add arc-level heterogeneity.
* **KPIs:** mean on-time %, p50/p95 on-time, mean tardiness, total distance, vehicles, and solver runtime.

You ran:

data/solutions\_ortools (DET)

data/solutions\_quantile/m1.2\_a0 (Q120)

data/solutions\_saa/k16\_b0p3 (SAA16-b0p3)

data/solutions\_saa/k32\_b0p5 (SAA32-b0p5)

data/solutions\_saa/k64\_b0p7 (SAA64-b0p7)

and evaluated them together with:

scripts/evaluate\_plans.py --K 200 --seed 42 --cv\_global 0.20 --cv\_link 0.10

→ data/reports/step8\_eval.csv

→ data/reports/step8\_eval\_by\_method.csv

Finally, you selected per-instance **champions** meeting a **99% on-time target**:

scripts/pick\_champions.py --target 99

→ data/reports/champions.csv

→ data/reports/champions\_stats\_by\_method.csv

→ data/champions/ (the JSON plans)

→ data/figures/champions\_cost\_vs\_ontime.png

**4) Key results**

**4.1 Why stochastic beats deterministic**

* **Deterministic plans** schedule tightly around nominal travel times. When we replay them with realistic variability, delays compound through the shift, and vehicles miss later windows. In your evaluation, DET’s *mean* on-time landed around the **~40%** range overall (box-plots by family show many instances between ~25–50%).
* **SAA and Q-buffer both pre-position slack** where it matters.
  + **Q-buffer** spreads time inflation uniformly, which is simple and **very effective**: in your tests it boosted on-time to **≈99–100%** across families with a **modest** distance increase (your scatter shows Q120 around **~99.5% on-time** at **~1.2×** DET distance on the champion set plot).
  + **SAA** is **smarter**: it *learns where uncertainty bites* (busy arcs, tight segments) and routes around risk hotspots. The **SAA16-b0p3** setting delivered **~99.7% on-time** with the **lowest mean distance among all champions**, often **below Q120** and with **near-baseline fleet size** thanks to the vehicle penalty.

**Bottom line:** deterministic planning can be brittle; optimizing *with* uncertainty (SAA) is consistently more reliable and, with the right settings, cheaper than uniform buffers.

**4.2 Cost vs. on-time trade-off (method level)**

Your figures show the method-level Pareto:

* **DET:** low cost (~lowest distance) but **poor reliability** (~40% on-time).
* **Q120:** **near-perfect reliability** (~99–100%) at **moderate cost** (+~8–12% vs DET on average).
* **SAA16-b0p3:** **near-perfect reliability** **with even lower cost than Q120**; this is the best “value” point.
* **SAA32-b0p5** and **SAA64-b0p7:** slightly **higher costs** (more slack, sometimes +1 vehicle) with **no material on-time gain** once the 99% target is met.

**4.3 Family-level behavior**

* **C** (clustered customers): easiest; Q120 and SAA hit 99–100% across the board.
* **R** and **RC** (random / mixed): hardest under DET (compounded delays); Q120 and SAA again push them into the 99–100% range.
* This confirms the uncertainty layer is doing what we expect: stressing non-clustered travel improves the signal for robust methods.

**5) Sensitivity to K (scenarios) and β (risk weight)**

* **K (16 → 64):** Increasing K raises compute and tends to **increase** cost slightly (because the optimizer “sees” more rare delay patterns and becomes more cautious). In your run, moving to **K=64, β=0.7** clearly **added distance** without visible on-time benefit beyond 99%.  
  **Guideline:** **K=16** is a great default for ~100-customer instances; use **K=32** if windows are extremely tight or you target **≥99.5%** on-time.
* **β (0.3 → 0.7):** Higher β penalizes tardiness more in the inner loop, pushing the search to add slack or vehicles. In your data:
  + **β=0.3** (SAA16-b0p3) provided the **best cost** while already achieving **≥99%** on-time.
  + **β=0.5 / 0.7** consumed extra cost with **diminishing returns** on reliability once above the 99% target.  
    **Guideline:** set **β=0.3** for cost-efficient ≥99%; increase β only if you must push on-time to **≥99.7%** and are willing to pay 5–10% extra distance or +1 vehicle on some instances.

**6) Practical guidance: when to use Q-buffer vs SAA**

* **Use Q-buffer (m=1.2)** when:
  + You need a **very fast**, **simple** robustification for *all* instances.
  + You can accept a **uniform** buffer (no per-route intelligence).
  + Runtime and engineering simplicity dominate.
* **Use SAA (K=16, β=0.3)** when:
  + You need **best cost** at **≥99%** on-time.
  + Some instances are **tight** or highly variable and deserve targeted slack.
  + You can afford a little extra compute (still minutes on Solomon-100 sizes).
* **Escalate** to **K=32** or **β≥0.5** only if required by SLA (>99.5%) or especially difficult windows; expect higher cost.

**7) What to include in the report (Step 9)**

**Files (keep under version control)**

* **Solutions (JSON):**  
  data/solutions\_ortools/, data/solutions\_quantile/m1.2\_a0/,  
  data/solutions\_saa/k16\_b0p3/, data/solutions\_saa/k32\_b0p5/, data/solutions\_saa/k64\_b0p7/.
* **Champions:**  
  data/champions/ (final picked plans),  
  data/reports/champions.csv, data/reports/champions\_stats\_by\_method.csv,  
  data/figures/champions\_cost\_vs\_ontime.png.
* **Evaluation CSVs:**  
  data/reports/step8\_eval.csv, data/reports/step8\_eval\_by\_method.csv.
* **Final table for appendix:**  
  data/reports/final\_table.csv, data/reports/final\_table\_pretty.xlsx  
  (generated by scripts/make\_final\_table.py after you fixed champions.csv—it now contains distance and vehicles).

**Figures to paste**

* **On-time by family & method** (box-plot).
* **Cost vs On-time (method level)** scatter.
* **Champions cost vs on-time** scatter (your latest plot).

**Text blocks (you can copy from this section)**

* Executive summary (first paragraph).
* Method & evaluation setup (Sections 2–3).
* Key results and sensitivity (Sections 4–5).
* Practical guidance (Section 6).

**8) Limitations and notes**

* The uncertainty model is *generic* (lognormal multipliers with moderate correlation). If you have historical travel times, you can refit CVs or time-of-day patterns.
* We optimize against **expected** cost and tardiness; extreme-risk guarantees (e.g., worst-case at 99.9%) would need a different robust model (e.g., Γ-robust, Step 10).
* Report both **strict on-time %** and **mean tardiness**—the latter shows residual lateness magnitude when it occurs.

**9) Reproducibility checklist (what you already did)**

1. Generated plans for DET, Q120, SAA16-b0p3, SAA32-b0p5, SAA64-b0p7.
2. Evaluated all plans with the same 200 scenarios (evaluate\_plans.py).
3. Picked per-instance champions at 99% target (pick\_champions.py).
4. Built the **final comparison table** (make\_final\_table.py) for the appendix.

**One-paragraph abstract (Step 9)**

*We implemented a stochastic planner for VRPTW using Sample Average Approximation (SAA) and compared it against a deterministic baseline and a fast quantile-buffer robustification on the 56 Solomon instances. All plans were evaluated with the same 200 correlated travel-time scenarios. The deterministic baseline achieved only ~40% on-time, confirming its brittleness under variability. The quantile method (1.2×) raised on-time to ≈99–100% with moderate cost growth. The SAA approach (K=16, β=0.3) achieved similar ≥99% on-time but with* ***lower distance*** *than the quantile method and fleet sizes close to baseline, thus offering the best cost-reliability trade-off. More conservative SAA settings added cost with limited additional reliability once above 99%. We recommend Q-buffer when speed and simplicity matter, and SAA (K=16, β=0.3) when minimizing cost at high service levels is the priority.*

**Final nits (so you’re 100% set)**

* Ensure data/reports/champions.csv **includes** distance and vehicles (your pick\_champions.py already writes those); then scripts/make\_final\_table.py will generate:
  + data/reports/final\_table.csv (per-instance cost, vehicles, on-time p50/p95, runtime),
  + data/reports/final\_table\_pretty.xlsx (nicely formatted for the appendix).
* Commit the whole data/champions/, data/reports/, and data/figures/ folders.

When you’re ready, we can move to Step 10 (Γ-robust) or straight to the final write-up polishing.