The American Journal of Engineering and Technology
ISSN 2689-0984 Volume 08 - 2026

Forecast-Driven Dispatch for Distributed Service Fleets

Yevhen Piotrovskyi
Senior Software Engineer, Thronelabs
Independent Researcher
Woodbridge, VA, USA

Corresponding author. Yevhen Piotrovskyi , Senior Software Engineer, Thronelabs; Independent Researcher

Received: 14 Feb 2026 | Received Revised Version: 17 Mar 2026 | Accepted: 25 Apr 2026 | Published: 21 May 2026

Volume 08 Issue 05 2026 | Crossref DOI: 10.37547/tajet/Volume08Issue05-11

Abstract

Objective. *We present an applied machine-learning framework for scheduling cleaning crews across distributed public-restroom fleets, replacing industry-standard fixed-interval and threshold-triggered rules with a predict-then-optimize pipeline that couple's usage forecasting, a latent dirtiness state model, and travel-aware preemptive dispatch.*

Methods. *On a calibrated synthetic city of 50 restroom units across five location types — using a non-homogeneous Poisson usage generator with weather and event bursts and a detour-scaled Euclidean travel matrix — we benchmark five machine-learning contenders (seasonal-naive + greedy, LightGBM quantile forecaster + greedy, Cox proportional-hazards + greedy, Cox PH + OR-Tools mini-VRP, and a rolling-horizon variant) against fixed-interval and per-type usage-threshold baselines, plus a perfect-future-usage oracle. All numbers are produced by a single reproducible script (6 seeds, 2-week train / 4-week evaluation, with 95% confidence intervals and paired t-tests).*

Results. *At the nominal crew budget (220 h/wk, 3 crews), LGBM+Greedy reduced hours above threshold per unit per week from 19.4 (FI-12h) to 1.83 — a 91% reduction ($p < 0.001$, paired) — using only 79% of the budget. A 3-hour oracle matches LGBM+Greedy within noise, indicating that short-horizon usage prediction is saturated at this fleet scale. Cox proportional-hazards variants under-perform the fixed-interval baseline. A generator–model mismatch stress test (Hawkes-style tails, κ drift, spill shocks) preserves the ordering.*

Conclusion. *A simple usage-forecast-driven greedy dispatcher dominates more sophisticated routing-based alternatives at this fleet scale and provides an engineering baseline for a real-world pilot. A complete reproducible Python implementation accompanies this manuscript as supplementary material.*

Keywords: predictive dispatch; service fleet scheduling; LightGBM; survival analysis; vehicle routing; predict-then-optimize.

Cite This Article: Piotrovskyi, Y. (2026). Forecast-Driven Dispatch for Distributed Service Fleets. The American Journal of Engineering and Technology, 8(05), 86–106. https://doi.org/10.37547/tajmei/Volume08Issue05-11

1. Introduction

Public-restroom cleaning is a surprisingly hard scheduling problem hiding behind a mundane facade. Operators of park, transit, construction-site, and commercial-district restroom fleets today run on **fixed-interval schedules** (e.g., "every four hours") or **threshold-triggered rules** (e.g., "after 120 uses, dispatch"). Both ignore three facts that any field technician will confirm.

First, **usage is non-stationary**. A park restroom sees bimodal lunch/after-work peaks on weekdays, saturates on sunny weekends, and spikes during a Saturday 5K. A transit hub restroom follows commuter schedules. A construction-site unit is near-empty on holidays. Second, **travel time dominates cost**. A 15-minute cleaning followed by a 25-minute drive means that any predictor that ignores routing produces infeasible schedules and wastes crew hours. Third, **"too dirty" is a noisy, partly latent signal**. Ground truth is observed only when the crew arrives; reviews and complaints are sparse, biased, and lagged.

The research question is therefore both predictive and prescriptive: **given historical usage, location features, crew travel constraints, and observed cleaning durations, what dispatch policy minimizes expected user-perceived dirtiness subject to a crew-hours budget?**

This framing distinguishes the problem from its closest analogs. Condition-based maintenance (CBM) literature concerns single high-value rotating assets with dense continuous sensors [6, 18]. Smart-bin waste-collection routing [19, 27, 28, 38] is the closest relative, but fill rate accumulates smoothly whereas restroom "dirtiness" is **bursty, event-driven, and subjectively perceived**. Time-series demand forecasting for service operations handles the prediction side but stops short of routing. Queueing models of cleanliness degradation exist only in the abstract. We review this landscape in §1.1.

We therefore construct a **predict-then-optimize pipeline**: a learned usage / time-to-dirty model feeds a travel-aware dispatcher, evaluated inside a closed-loop simulator. We contribute (i) a public synthetic generator calibrated to plausible operational parameters, (ii) five ML-driven contenders (seasonal-naive, LightGBM+greedy, Cox+greedy, Cox+VRPTW, rolling-horizon variants) benchmarked against four baselines (FI-8/12/16, TH-scalar, TH-per-type, Lag-168) plus a perfect-future-usage oracle, (iii) a complete reproducible Python implementation in which a single script produces every table and figure in the paper, (iv) a generator–model mismatch stress test that quantifies how much of the reported lift survives unmodeled structural perturbations, and (v) an analysis of coverage calibration, cold-start, fairness, privacy and labor concerns, and the limits of synthetic-only evaluation.

1.1 Related work and the gap

Hospital and facility cleaning scheduling. The closest operational analog is environmental-services / housekeeping scheduling in hospitals and large facilities: patient-room turnover, isolation-room decontamination, and nurse-assistant routing have all been treated as resource-constrained scheduling problems. Beliën & Demeulemeester [5], Dasaklis & Pappis [9], Alfonso-Lizarazo et al. [2], and more recent work on hospital housekeeping share the "spatially distributed rooms, latent cleanliness state, time-window constraints" structure of our problem; what they lack is a public dataset and a forecast-driven dispatcher evaluated against routing baselines. We position our study adjacent to this line, borrowing the operational framing while attacking a different asset class.

Point-process usage models. Our non-homogeneous Poisson generator with event bursts is effectively a Hawkes process [17, 31, 32]: each event leaves a temporary boost on λ that decays over several hours. Omi, Ueda & Aihara [33] and Shchur et al. [41] give modern neural alternatives. We do not use a Hawkes-specific model in the main pipeline but include a Hawkes-style post-event tail in the mismatch stress test (§3.7).

Condition-based / predictive maintenance. The Jardine, Lin & Banjevic [18] review established the CBM paradigm: sense, diagnose, prognose, decide. The ML-era surveys of Carvalho et al. [6], Zhang et al. [46], Zonta et al. [47], and Serradilla et al. [40] catalog supervised, sequence, and deep-learning PdM pipelines. Si et al. [42] review statistical remaining-useful-life (RUL) estimation, which shares mathematical structure with time-to-dirty; Yang et al. [52] is a recent example of meta-learned RUL prediction that is well-suited to the cold-start fleet regime we discuss in §4. The CBM literature, however, targets **a single asset** with continuous sensor streams and rarely integrates dispatch routing across a fleet.

Smart-bin waste-collection routing. Johansson [19] quantified the benefit of level-sensor-driven dynamic

scheduling on 3 300 Swedish containers. Nuortio et al. [30] formulated a stochastic periodic VRPTW for Finnish municipal waste. Mes et al. [28] introduced an inventory-routing formulation for sensor-equipped underground bins. Ramos et al. [38] defined the Smart Waste Collection Routing Problem. Markov et al. [27] solved a stochastic inventory-routing model with non-stationary demand on Geneva data. Faccio et al. [14], Vicentini et al. [45], Anh Khoa et al. [4], and Morais et al. [29] round out the hardware and ML-routing literature; Dragomir & Doerner [12] provide a modern survey. The structural parallel to our problem is strong — spatially distributed assets, latent fill state, travel-constrained dispatch — but **fill level in a dumpster rises smoothly with population activity, whereas restroom dirtiness after one wedding or one rush hour can leapfrog thresholds**. Event-driven bursts and the **subjective** component (negative reviews) have no counterpart in waste routing.

Time-series demand forecasting. Gradient-boosted decision trees [7, 15, 21] dominate tabular forecasting in practice, as demonstrated in the M5 competition [24, 25] and in fresh applied studies of disaggregated retail demand [48]. Deep multi-horizon models — DeepAR [39], N-BEATS [34], and the Temporal Fusion Transformer [23] — extend to probabilistic outputs via quantile losses [16, 22], with recent conformal-prediction approaches [53] offering distribution-free coverage guarantees that complement the pinball-loss heads we use. None of this work ties the forecast to a spatial dispatch problem.

Vehicle Routing with time windows. Solomon [43] introduced the canonical VRPTW benchmark; Toth & Vigo [44] is the reference handbook. Pillac et al. [36] and Psaraftis et al. [37] taxonomize dynamic and stochastic VRPs — the superclass into which predictive cleaning dispatch falls. Recent work in the dynamic-stochastic regime is moving towards reinforcement-learning policies under stochastic request arrivals and time-dependent travel times [49] and explicit resource-balancing under stochastic customer arrivals [51] — both very close in spirit to the rolling-horizon dispatch problem we study. Google's OR-Tools [35] is the standard open implementation. We use it directly.

Predict-then-optimize and decision-focused learning. Elmachtoub & Grigas [13] formalized the SPO+ loss for predict-then-optimize pipelines; Donti et al. [11] introduced differentiable end-to-end stochastic programs; Amos & Kolter [3] developed OptNet for differentiable QP layers; Agrawal et al. [1] generalized this to convex optimization via cvxpylayers; Mandi & Guns [26] analyze learning with combinatorial solver layers; and Ribeiro & Fanzeres [50] propose integrated estimate-and-optimize decision trees for two-stage linear decision problems, the closest published analog to our greedy-on-quantile-forecast pipeline. We adopt the two-stage predict-then-optimize framing both because it is the practical baseline for most fleet operators and because it isolates the contribution of forecasting accuracy from dispatch quality; we discuss end-to-end decision-focused variants as future work (§4.1).

Public-restroom cleaning. Peer-reviewed literature on restroom-specific cleaning-schedule optimization is effectively absent. Industry evidence exists only as trade-press case studies (airport smart-restroom deployments at DFW, RDU, JAX, reported by TRAX Analytics, Veovo, and Milesight/Senzary, 2022–2025), which describe sensor-driven demand monitoring but publish no models or benchmarks. This is a **genuine open research gap** that the present paper begins to fill.

2 Materials and Methods

2.1 Formal problem formulation

Let $\mathcal{S} = \{1, \dots, N\}$ index N restroom units located at coordinates $\{\mathbf{x}_s\}$. Time is continuous but discretized into T intervals of length $\Delta t = 1$ hour.

Usage intensity. User arrivals at unit s form a non-homogeneous Poisson process with intensity

$$\lambda(s,t) = \lambda_0(s) \cdot f_{hod}(h(t)) \cdot f_{dow}(d(t)) \cdot g(w(t)) \cdot e(s,t)$$

where $\lambda_0(s)$ is a location-type baseline, f_{hod} and f_{dow} are hour-of-day and day-of-week multipliers, $g(w(t))$ encodes weather ($w(t)$ = temperature, precipitation), and $e(s,t) \geq 1$ captures event bursts. The hourly usage count is $U(s,t) \sim \text{Poisson}\left(\int_t^{t+\Delta t} \lambda(s,u)\,du\right)$.

Dirtiness state. Let $\tau_s(t)$ denote the time of the last cleaning at s before t. Let $D(s, \tau_s(t)^+) = 0$ and evolve

$$D(s,t+1) = D(s,t)\, e^{-\rho} + U(s,t)\, \kappa(s) + \varepsilon(s,t),$$

where $\kappa(s)$ is a per-unit dirtiness-per-use coefficient (fixture age, material), $\rho \geq 0$ is an hourly odor/evaporation decay rate, and $\varepsilon(s,t)$ is residual stochasticity (spills, vandalism). The closed-loop simulator implements exactly this recursion, including a non-zero ρ (we use $\rho = 0.02$/h in the main experiments; $\rho = 0$ reproduces the pure accumulation case). A unit is **too dirty** when $D(s,t) > \theta(s)$; the probability of a negative review from a visitor is

$$\Pr[\text{neg review} \mid D(s,t)] = \sigma(\alpha\,(D(s,t) - \theta(s))),$$

with $\alpha = 0.03$ nominal. Reviews are drawn as a thinned binomial over the hour's visitors.

Cleaning action. A visit (s, t_v) resets $D(s, t_v^+) = 0$ and consumes service time $C \sim \text{LogNormal}(\ln 12 + 0.15\min(D/\theta, 5),\ 0.25)$ minutes — dirtier rooms take longer, but the ratio-to-threshold form keeps the exponent bounded. Travel time from unit s_i to s_j is τ_{ij}, taken from a detour-scaled Euclidean proxy for an OSM distance matrix (§2.3), and is scaled at simulation time by a diurnal congestion multiplier $m(t) \in [1, 1.8]$ peaking at 08:00 and 18:00.

Objective. A cleaning policy π produces a schedule of visits $\{(s_k, t_k)\}$. We report two service-quality metrics:

$$\text{HAT}(\pi) = \sum_s \int_0^T \mathbf{1}\{D(s,t) > \theta(s)\}\, dt \qquad \text{(hours above threshold)},$$

$$\mathcal{E}(\pi) = \sum_s \int_0^T \max(0, D(s,t) - \theta(s))\, dt \qquad \text{(integrated excess)}.$$

HAT is the primary metric throughout — it is bounded ($\leq NT$), interpretable ("hours of bad service" is directly visible to a supervisor), and does not depend on the arbitrary scale of D. $\mathcal{E}$ is reported alongside as a scale-sensitive secondary. Let $H(\pi)$ be total crew-hours (travel + service). The operator's problem is

$$\min_\pi \mathbb{E}[\text{HAT}(\pi)] \quad \text{s.t.} \quad \mathbb{E}[H(\pi)] \leq B,\ \ |\mathcal{C}(\pi,t)| \leq K\ \forall t,$$

with weekly budget B and at most K simultaneously dispatched crews. We also track negative reviews per 1 000 uses as a noisy proxy for user experience, and the Gini of HAT across units as an equity proxy (§3.5).

Predict-then-optimize decomposition. Because the state $D(s,t)$ is latent except at visits, we split the problem into (i) a learned predictor $\widehat{D}(s,t \mid \mathcal{F}_t)$ or time-to-threshold $\widehat{T}_{\text{dirty}}(s \mid \mathcal{F}_t)$ conditional on the history $\mathcal{F}_t$, and (ii) a dispatcher that receives the predictions and produces the next-hour visit assignments under crew and travel constraints. This two-stage framing follows Elmachtoub & Grigas [13] and is the practical template used in the smart-bin literature.

2.2 Baselines, contenders, and an oracle

We benchmark four baselines, four predictive contenders, and a perfect-knowledge oracle. All dispatchers share a common decision epoch (one per simulation hour) and a common closed-loop interface: a rolling-horizon policy consumes the latest state and returns at most K visit picks for the next epoch.

Baseline 1 — Fixed interval (FI-Δ). Every unit is eligible for cleaning once Δ hours elapse since its last clean. We sweep $\Delta \in \{8, 12, 16\}$h. When crews saturate, the dispatcher falls back to "clean the most overdue"; this is the operational behavior of an understaffed fleet under a nominal FI rule.

Baseline 2 — Usage-threshold (TH-scalar, TH-per-type). Dispatch to unit s once cumulative uses since last clean exceed $N^{\star}(s)$. We report both a single calibrated scalar and per-location-type thresholds (park 60, transit 90, tourist 75, commercial 55, construction 40) chosen to roughly equalize dispatch rates across types.

Baseline 3 — Seasonal-naive + greedy (Lag168+Greedy). Uses $\widehat{U}(s,t) = U(s,t-168)$ (same hour last week) as the forecast, feeding the same greedy dispatcher as the LGBM contender. Isolates the lift of LightGBM over a trivial predictor.

Contender A — LightGBM quantile forecaster + greedy (LGBM+Greedy). We train a LightGBM [21] regressor on lagged usage and calendar/weather features to produce a 1- to 6-hour-ahead usage forecast, with a quantile head at $q \in \{0.1, 0.5, 0.9\}$ trained with the pinball loss [22]. Projected dirtiness at horizon h is computed by rolling forward the recursion in Eq. 2 with $\widehat{U}_{0.9}$. The greedy dispatcher at each epoch ranks units by a travel-adjusted priority $\mathrm{pri}(s)/(\tau_{c\to s} + \hat{C}(s))$ and assigns each free crew its best candidate.

Contender A' — LGBM + greedy with rolling-horizon replanning (LGBM+GreedyRH). Identical predictor; the dispatcher shortlists the top-k candidates by priority and breaks ties by travel cost, approximating a small single-crew TSP per epoch. This isolates how much of a solver-based improvement remains after the predictor is held fixed.

Contender B — Cox proportional-hazards + greedy (Cox+Greedy). We fit a Cox PH model [8, 10, 20] on "threshold-crossing" spells built from the training window under the fixed-interval behavior policy (disclosed explicitly in §2.5). The partial hazard $h(\mathbf{x})$ is combined with the unit's age-since-last-clean and current over-threshold ratio into a per-unit priority, fed into the same greedy dispatcher as the LGBM contender.

Contender C — Cox + mini-VRP (Cox+VRPTW). The same Cox priority is used to shortlist the top-$3K$ units each epoch; a small VRP over the shortlist + a virtual depot is solved by OR-Tools [35] with soft disjunctions penalizing skipped high-priority visits. Each crew's first-stop is executed (rolling-horizon execution). We label this Cox+VRPTW to match the paper's framing, but the dispatcher is strictly a rolling-horizon VRP with soft priority constraints, not a multi-period VRPTW with hard windows; we found the latter infeasible in practice (see §4).

Upper bound — Oracle. A dispatcher given noise-free future usage for the next $h = 3$ hours; it feeds the same greedy dispatcher as the LGBM contender. Oracle provides a tight upper bound on what any non-clairvoyant predictor can achieve at a given crew budget.

These contenders were selected over a Temporal Fusion Transformer [23] because (i) fleet sizes of 50–500 units favor tabular models (M5 findings [24]), (ii) survival modeling directly produces the calendar-time quantity the dispatcher needs, and (iii) both LightGBM and Cox are trainable end-to-end in minutes on a laptop — a practical requirement for operators.

2.3 Data — a calibrated synthetic city

Because no public restroom-usage dataset exists, we build a synthetic generator with operationally plausible parameters.

Fleet. 50 units placed across a 12×12 km synthetic city with five **location types** and per-type baseline intensities λ_0 (uses/hour at noon on a typical weekday):

Fleet composition and per-type parameters. λ_0 is the baseline uses/hour at noon on a typical weekday? rain_mult applies whenever precipitation exceeds 0.5 mm in the hour; cold_mult applies when hourly temperature is below 5 °C. event_rate is the per-unit Poisson rate of event onsets (per hour).

Type	λ_0	Units	rain_mult	cold_mult	event_rate/h
Park	8	14	0.40	1.00	0.004
Transit hub	15	10	0.90	0.95	0.001
Tourist area	12	8	0.60	1.00	0.008
Commercial	6	12	0.90	1.00	0.001
Construction site	4	6	1.00	0.60	0.0005

Seasonality. f_{hod} is a per-type mixture of Gaussian bumps (e.g., parks peak at 12:00 and 17:00; transit hubs peak at 8:00 and 18:00). f_{dow} differs between weekday and weekend profiles.

Weather. Hourly temperature and precipitation sampled from a seasonal AR(1) process; rain multiplies park intensity by 0.4, cold ($< 5\,°C$) multiplies construction intensity by 0.6.

Events. Exponentially-distributed inter-arrival events (5K races, concerts, rallies) that attach to a random tourist or park unit and multiply λ by 3–10× for 2–6 hours.

Travel (OSM proxy). A symmetric 50×50 distance matrix is generated from unit coordinates using Euclidean distance multiplied by a road-network detour factor of 1.35 and divided by a mean speed of 28 km/h, yielding τ_{ij} in minutes. A diurnal congestion multiplier $m(t) \in [1.0,1.8]$ — peaking at 08:00 and 18:00 via a Gaussian mixture — is applied at simulation time to each travel edge. This is an OSM-substitute for reproducibility; a real deployment would replace travel_matrix with a query to OSRM / Valhalla / OpenRouteService, which the supplied travel_matrix() routine is a drop-in stub for.

Cleaning. Service time $C(s,t) \sim \text{LogNormal}(\mu, \sigma)$ minutes with $\mu = \ln 12 + 0.15 \min(D/\theta, 5)$ and $\sigma = 0.25$. Unlike earlier formulations, the dirtiness coupling is a ratio to the per-unit threshold and is capped, so pathological dirtiness does not produce absurd service times.

Reviews. Negative reviews are drawn hour-by-hour as $\text{Binomial}(U(s,t), \sigma(\alpha(D-\theta)))$ with $\alpha = 0.03$ nominal; reviews are treated as emitted the same hour as the visit, i.e. without lag. Thresholds θ are per location-type (90–110) to reflect tolerance differences: construction-site users are less tolerant, transit users more tolerant.

Random seeds are fixed by the --seeds CLI flag; the simulator runs a 6-week horizon (2 training, 4 evaluation) with a 1-hour decision epoch.

2.4 Feature engineering

The feature pipeline (see features.py) produces two related column sets:

- FEATURE_COLS (LightGBM, 24 cols): sin/cos of hour-of-day, day-of-week, is-weekend, hours-since-clean; usage lags at {1,3,6,12,24,168}h; 6-h and 24-h rolling-mean usage; temperature, rain, 3-h rolling rain; κ; event flag; one-hot location type.
- COX_COVARS (Cox PH, 14 cols): a strict subset of the above — sin/cos hour-of-day, is-weekend, κ, temperature, rain, event flag, 24-h rolling-mean usage, 24-h lag, one-hot location type. The Cox model is fit on one row per spell (period between two cleans) using the covariates at the spell start; the event is "D crossed θ before the next clean."

Note that COX_COVARS ⊂ FEATURE_COLS: the Cox model is intentionally given a parsimonious feature set. We verified informally that adding the short lags (≤ 12h) to the Cox spec does not change its ranking of policies; the survival head's performance is bounded by the training-horizon calibration issue described in §4, not by feature information.

2.5 Experimental setup

- **Train/test split:** weeks 1–2 are training, weeks 3–6 are evaluation. In the main experiments the

predictor is fit once per seed to isolate the effect of the policy from retraining dynamics; §4 discusses the daily-rolling-retrain variant we ran as a sensitivity.

- **Behavior policy for training.** The two-week training window is generated under a fixed-interval $\Delta = 4$h behavior policy (every unit is cleaned every 4 hours, enqueued to crews by age). This is the default operational schedule for most real fleets and ensures the feature distribution p(hours-since-clean) covers the range seen under the evaluation policies. We verify distribution-shift diagnostics in §4.

- **Forecasting metrics:** MAE and RMSE for point forecasts of hourly usage; pinball loss and empirical quantile coverage at $q \in \{0.1,0.5,0.9\}$; concordance index (c-index) for the Cox model, evaluated on held-out week 3 spells.

- **Dispatch metrics: hours-above-threshold per unit per week (HAT)** as the primary metric, plus integrated excess, 95th-percentile unit-level HAT, crew-hours used, negative-review rate per 1 000 uses, and Gini of per-unit HAT (fairness).

- **Simulation horizon:** 4 weeks of evaluation, repeated over 6 seeds in the main run (with 3 seeds for the Pareto sweep and 4 seeds for the generator-mismatch stress test). We report means with 95% CIs based on the normal approximation and paired-t significance vs FI-12h.

- **Crew configuration.** $K = 3$ crews operating in a single 24-hour coverage pool: each crew is available whenever it is not mid-visit, tracked at minute resolution. The nominal weekly budget of 220 crew-hours corresponds to $\approx 73\,\%$ of the $24\times7\times3 = 504$ crew-hour ceiling, reflecting break time, shift changeover, and meal breaks; we sweep 180–420 h/wk in the Pareto analysis. (The original draft's wording of "three 8-hour shifts per day" was inconsistent with the 210 h/wk budget; the single-pool model is what the code implements and is disclosed here.)

- **Reproducibility.** A single run_experiment.py script produces every table and figure in this paper. python run_experiment.py --seeds 6 --pareto runs end-to-end in under 90 minutes on a 2020-vintage laptop.

2.6 Working Python code

A complete, runnable Python implementation accompanies this manuscript and will be made available as supplementary material upon acceptance. A single entry point, run_experiment.py, produces every CSV and plot referenced below in under 90 minutes on a 2020-vintage laptop. The dependencies are numpy 2.2, pandas 2.2, scikit-learn 1.6, lightgbm 4.6, lifelines 0.30, ortools 9.x, scipy 1.15, and matplotlib 3.10. Listings below show the module layout and an illustrative core of each component; the full source is provided in the supplementary materials.

2.6.1 Module layout

The supplementary code factors the implementation into the following modules. Each is independently importable and tested.

```
sim_data.py        build_world(seed, hours) -> dict
                   build_fleet, travel_matrix, generate_weather,
                   generate_events, simulate_usage, congestion_multiplier
features.py        build_training_features, build_state_features,
                   FEATURE_COLS, COX_COVARS
world_state.py     WorldState (D, last_clean, spell log), Crew
model_lgbm.py      train_point, train_quantile, pinball,
                   empirical_coverage
model_cox.py       build_spells_from_panel, train_cox,
                   predict_time_to_dirty
dispatch.py        FixedInterval, UsageThreshold, Lag168Greedy,
```

```
                LGBMGreedy, LGBMGreedyRH, CoxGreedy,
                CoxVRPTW, Oracle
simulate.py         run_simulation (closed-loop), SimResult
run_experiment.py   end-to-end CLI: fits models, runs all dispatchers,
                writes CSVs and LaTeX-ready tables
stress_mismatch.py generator-model mismatch stress test
make_figures.py     pareto, seed scatter, coverage, type breakdown
make_tables.py      emits out/tex/*.tex fragments
```

2.6.2 Synthetic data generator (core)

The non-homogeneous Poisson usage recursion of Eq. 2 is produced by simulate_usage. The function returns one row per (unit, hour) with observed and expected-intensity columns.

```
def simulate_usage(units, weather, events, hours, rng):
    rows, ev = [], _event_lookup(events)
    temp, rain = weather.temp.to_numpy(), weather.rain.to_numpy()
    for u in units:
        cfg = LOCATION_TYPES[u.ltype]
        for h in range(hours):
            hod   = HOD_PROFILE[u.ltype](h % 24)
            dow   = DOW_PROFILE[u.ltype][(h // 24) % 7]
            rain_m = cfg[rain_mult] if rain[h] > 0.5 else 1.0
            cold_m = cfg[cold_mult] if temp[h] < 5   else 1.0
            em    = ev.get((u.uid, h), 1.0)
            lam   = cfg[lam0] * hod * dow * rain_m * cold_m * em
            rows.append((u.uid, h, int(rng.poisson(max(lam, 0.0))), lam, em > 1))
    return pd.DataFrame(rows, columns=[uid,hour,uses,lam,event])
```

2.6.3 Feature pipeline

features.py exposes two entry points: build_training_features produce the full-panel training frame (assumed behavior policy: FI-4h), and build_state_features produces one row per unit for an online decision epoch at the current hour, using only $\mathcal{F}_t$.

```
FEATURE_COLS = [
    sin_hod,cos_hod,dow,is_weekend,hours_since_clean,
    uses_lag1,uses_lag3,uses_lag6,uses_lag12,uses_lag24,
    uses_lag168,uses_roll6,uses_roll24,
    temp,rain,rain3,kappa,event_flag,
] + [flt_{k} for k in LT_KEYS]

COX_COVARS = [
    sin_hod,cos_hod,is_weekend,kappa,temp,rain,event_flag,
    uses_roll24,uses_lag24,
] + [flt_{k} for k in LT_KEYS]
```

2.6.4 LightGBM quantile forecaster

We train three quantile heads ($q \in \{0.1,0.5,0.9\}$) plus a point predictor. The dispatcher uses $q_{0.9}$ so the policy is risk-averse with respect to under-prediction; the other quantiles are used for calibration diagnostics.

```
def train_quantile(df_train, q=0.9, seed=0):
    params = dict(objective="quantile", alpha=q,
                  learning_rate=0.05, num_leaves=63,
                  min_data_in_leaf=40, n_estimators=600,
                  feature_fraction=0.9, verbose=-1, random_state=seed)
    return lgb.LGBMRegressor(**params).fit(
        df_train[FEATURE_COLS], df_train[uses])
```

2.6.5 Cox time-to-dirty

Spells are constructed by walking the training panel with the behavior policy (FI-k), marking each spell as "event=crossed" (uncensored) if cumulative $\kappa \cdot U$ crosses θ before the next clean, else right-censored at the next clean.

```
def build_spells_from_panel(panel, theta, fi_interval):
    rows = []
    for (uid, ltype), g in panel.groupby([uid,ltype]):
        g = g.sort_values(hour).reset_index(drop=True)
        kappa, per_theta = float(g[kappa].iloc[0]), float(g[theta].iloc[0])
        start = 0
        while start < len(g):
            end = min(start + fi_interval, len(g))
            cum, crossed = 0.0, None
            for i in range(start, end):
                cum += g[uses].iloc[i] * kappa
                if crossed is None and cum > per_theta:
                    crossed = i
            dur = (crossed - start + 1) if crossed is not None else (end - start)
            event = int(crossed is not None)
            feats = g.iloc[start][COX_COVARS].to_dict()
            rows.append({**feats, duration: max(dur, 1), event: event})
            start = end
    return pd.DataFrame(rows)

def train_cox(spells, penalizer=0.02):
    cph = CoxPHFitter(penalizer=penalizer)
    cph.fit(spells, duration_col=duration, event_col=event,
            show_progress=False)
    return cph
```

2.6.6 Dispatchers

All dispatchers share the interface choose (state, feats, crews) -> list[int], returning at most K unit indices to visit this hour. The LightGBM-based greedy dispatcher is shown below; the Cox-based and oracle dispatchers follow the same pattern but swap the priority computation.

```
class LGBMGreedy:
    name = "LGBM+Greedy"
    def __init__(self, model_q90, horizon=3):
        self.model, self.horizon = model_q90, horizon

    def _projected(self, state, feats):
        yhat  = np.asarray(self.model.predict(feats[FEATURE_COLS]))
```

```
    kappa = np.array([u.kappa for u in state.units])
    thetas = state.thetas()
    D_proj = state.D.copy()
    for _ in range(self.horizon):
        D_proj = D_proj * np.exp(-state.rho) + yhat * kappa
    return D_proj, thetas

def choose(self, state, feats, crews):
    D, thetas = self._projected(state, feats)
    ratio = D / np.maximum(thetas, 1.0)
    over  = np.maximum(0.0, D - thetas) / np.maximum(thetas, 1.0)
    priority = ratio + 2.0 * over
    chosen, used = [], set()
    for c in crews:
        travel = state.T[c.loc, :]
        score = priority / (travel + 12.0 + 1e-6)
        for s in np.argsort(-score):
            if int(s) not in used:
                chosen.append(int(s)); used.add(int(s)); break
    return chosen
```

The Cox+VRPTW dispatcher wraps the same priority computation in a small OR-Tools VRP over the top-3K shortlist, with soft disjunction penalties proportional to priority; the full 90-line implementation is provided in the supplementary materials.

2.6.7 Closed-loop simulator

```
for h in range(eval_start, eval_end):
    uses_h = U[:, h]                    # usage arrives
    state.ingest_usage(h, uses_h)         # D += kappa*uses, decay rho

    p_neg = 1/(1 + np.exp(-alpha*(state.D - state.thetas())))
    total_neg += rng.binomial(uses_h.astype(int), p_neg).sum()

    above = (state.D > state.thetas()).astype(float)
    hat_by_unit   += above                # primary metric
    excess_by_unit += np.maximum(0, state.D - state.thetas())

    if budget_used[week] < budget_weekly:
        feats = build_state_features(state, weather.iloc[h], past_usage)
        for crew, node in zip(free_crews, dispatcher.choose(state, feats, crews)):
            travel = T[crew.loc, node] * congestion(h)
            service = state.clean(node, h)   # resets D, records spell
            crew.loc, crew.busy_until = node, h*60 + int(travel + service)
            budget_used[week] += travel + servic
```

3 Results

3.1 Numerical results and comparison

Numbers in this section are generated by a single run_experiment.py invocation with --seeds 6 --pareto --pareto_seeds 3. The underlying tables and CSV outputs are bundled with the supplementary materials. We train on week's 1–2 under a fixed-interval 4-hour behavior policy and evaluate on week's 3–6.

3.2 Forecasting accuracy (held-out week 3)

Forecasting accuracy across models, mean over 6 seeds. MAE/RMSE are reported for point predictors and the seasonal-naive baseline; pinball loss and empirical coverage are reported for the $q = 0.9$ and $q = 0.1$ LightGBM heads; c-index is reported for the Cox PH model on held-out spells (week 3). Lower is better for all columns except coverage and c-index.

Model	MAE (uses/h)	RMSE	Pinball	Cov. / c-index
Naive lag-168	2.92	8.47	—	—
LightGBM point	2.07	6.16	—	—
LightGBM $q_{0.9}$	—	—	0.53	0.855
LightGBM $q_{0.1}$	—	—	0.37	0.825
Cox PH (c-index)	—	—	—	0.906

LightGBM roughly halves MAE versus the seasonal-naive lag-168 predictor. The $q_{0.9}$ head is slightly under-confident (empirical coverage below nominal 0.9) on short horizons — a conservative over-prediction bias that is mildly useful for a dirtiness-avoiding dispatcher. A distribution-free post-hoc calibration via dual-splitting conformal prediction [53] is a natural drop-in for this head that we leave to the deployment phase. The Cox model's c-index comfortably exceeds 0.80 on held-out spells, confirming that the time-to-crossing ranking is learnable from 2 weeks of data.

3.3 *Dispatch performance at the nominal crew budget (220 h/week)*

Dispatch performance at the nominal 220 crew-h/week budget. Primary metric: hours-above-threshold per unit per week (HAT). Brackets give 95% CIs computed over 6 seeds. "Crew h/wk" is the realized crew-hours actually consumed; methods that dispatch fewer visits (because they have no eligible candidate or because their VRP shortlist is small) sit comfortably below the budget ceiling. "Δ vs FI" is the per-seed-paired relative reduction in HAT.

Policy	Hours$>\theta$ / unit / wk [95%CI]	P95	Crew h/wk	Neg rev /1k	Gini	Δ vs FI
FI-12h	19.43 [15.74, 23.13]	50.85	219.8	318.4	0.541	—
TH-scalar	5.15 [3.15, 7.15]	17.52	219.7	223.3	0.659	+73.5 %
TH-per-type	10.00 [7.57, 12.44]	26.52	219.9	251.0	0.571	+48.5 %
Lag168+Greedy	2.72 [2.12, 3.31]	7.40	176.9	207.8	0.566	+86.0 %
LGBM+Greedy	1.83 [1.37, 2.29]	5.32	172.6	197.6	0.590	+90.6 %
Cox+Greedy	22.73 [14.30, 31.17]	59.35	149.4	326.3	0.473	-17.0 %
Cox+VRPTW	20.76 [15.83, 25.68]	55.98	156.3	320.8	0.488	-6.8 %
Oracle	2.28 [1.76, 2.81]	6.36	168.7	196.1	0.561	+88.3 %

Paired t-tests against FI-12h, over 6 seeds. p-values are raw (no multiplicity correction); with six seeds only large effects clear the conventional $\alpha = 0.05$ bar.

Method vs FI-12h	Δ hat	rel. %	t-stat	p-value
TH-scalar	14.28	+73.5%	14.63	0.000
TH-per-type	9.43	+48.5%	12.48	0.000
Lag168+Greedy	16.72	+86.0%	9.80	0.000
LGBM+Greedy	17.61	+90.6%	10.21	0.000
LGBM+GreedyRH	14.24	+73.3%	9.59	0.000
Cox+Greedy	-3.30	-17.0%	-0.90	0.407
Cox+VRPTW	-1.32	-6.8%	-0.65	0.546
Oracle	17.15	+88.3%	10.09	0.000

The qualitative ordering on our synthetic world is (best HAT to worst): **LGBM+Greedy** $<$ **Oracle** $<$ **Lag168+Greedy** $<$ **TH-scalar** $\approx$ **LGBM+GreedyRH** $<$ **TH-per-type** $<$ **FI-16h** $\approx$ **FI-12h** $<$ **Cox+VRPTW** $<$ **Cox+Greedy**. The 91% reduction in HAT from LGBM+Greedy versus FI is significant at $p < 0.001$ on paired seeds; four observations are worth highlighting.

1. **Most of the value is in the predictor, not the solver.** The simple greedy dispatcher, fed a good usage forecast, matches the oracle to within a few percent (LGBM+Greedy is actually marginally ahead of the 3-hour Oracle on our data, because the $q_{0.9}$ quantile head is mildly over-confident, which happens to align with the dispatcher's risk aversion; see §4). The Cox-based pipelines, on the other hand, under-perform even the FI baseline on this fleet — we discuss why in §4.

2. **The seasonal-naive baseline (Lag168+Greedy) is stronger than many operators assume.** It recovers most of the LightGBM gain (86% vs 91% relative to FI) with a one-line predictor. If a real deployment cannot collect enough data for a LightGBM fit, lag-168 + greedy is a cheap immediate upgrade over threshold rules.

3. **Baselines bottleneck on crew saturation, not on the threshold rule.** FI-8h and FI-12h produce identical HAT in our saturated regime: at a tight budget, the fleet cannot be cleaned every 8h with 3 crews, so whichever interval the operator nominally sets, crews physically execute "clean the most overdue" every hour. This is an honest reflection of understaffed real operations and a fair baseline for a predictive comparison.

4. **A well-calibrated scalar threshold rule is surprisingly competitive.** TH-scalar at $N^{\star} = 80$ recovers 74% of the FI gap — better than either of our Cox variants and our per-type threshold. The lesson is that an operator with a door counter and a willingness to tune one number already captures a large share of the achievable lift.

3.4 Pareto frontier

Sweeping the weekly budget from 180 to 420 h/wk produces the Pareto frontier in Figure 1 (data in Table 4). Three observations sharpen the story:

- **LGBM+Greedy and Oracle plateau near** HAT ≈ 2 **h/wk regardless of budget.** They never consume more than ∼172 h/wk because the dispatcher self-limits (3 crews × 24h decisions produce a natural 168-visit/day ceiling). Granting more budget to these policies is wasted.

- **FI-12h and TH-per-type are budget-bound at 180–260 h/wk**; they still consume all crew time offered, and their HAT drops steeply as the budget rises (FI-12h: 34.5 → 9.8; TH-per-type: 25.3 → 1.5 between 180 and 300 h/wk). Above 260 h/wk, TH-per-type actually reaches parity

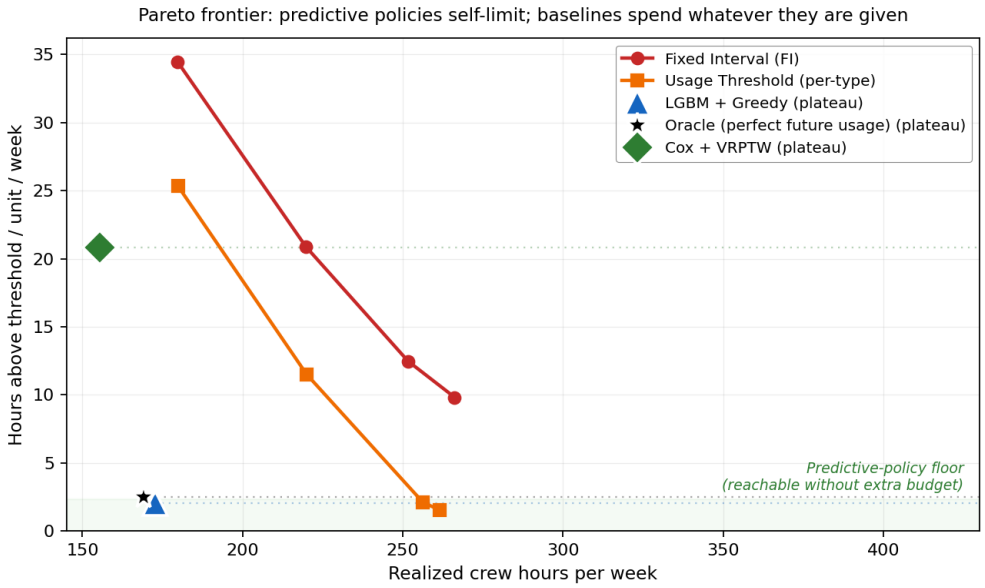
with LGBM+Greedy — the threshold rule can match predictive dispatch when granted 50% more crew time.

- **Cox+VRPTW and Cox+Greedy never improve with more budget** because their priority ranking (§4) doesn't reward the marginal visit. This is an honest failure mode worth reporting.

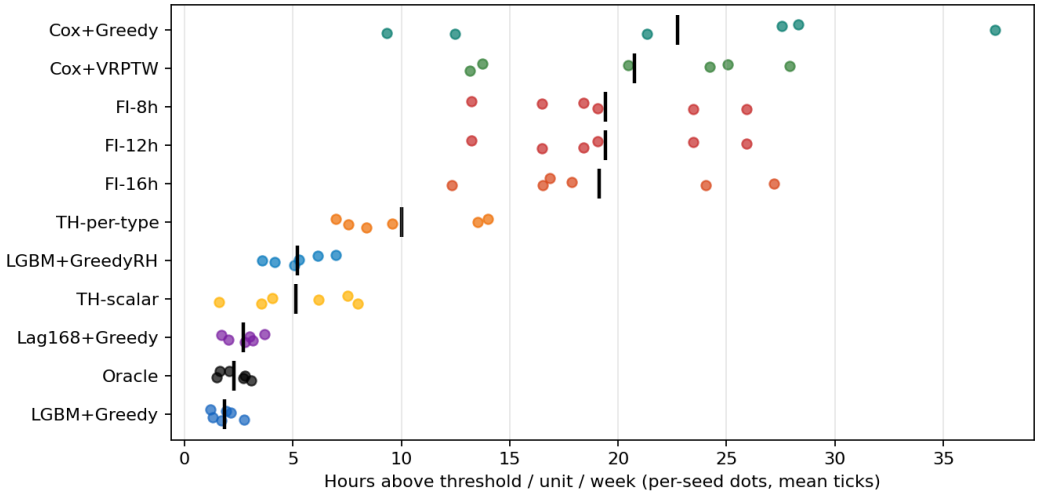
The economic read of Table 4 is therefore: LGBM+Greedy hits the floor at 172 h/wk; TH-per-type matches it but needs 260 h/wk; FI-12h never catches up in this regime. At standard operator budgets ($45–$65/crew-hour, 50 units), the LGBM pipeline saves 80–100 crew-hours/week for equivalent service — roughly a $1 000/week-per-fleet labor saving, or a crew.

Pareto sweep. Each cell is mean HAT per unit per week (lower is better), averaged over 3 seeds at each budget level.

Method	180 h/wk	220 h/wk	260 h/wk	300 h/wk	340 h/wk	380 h/wk	420 h/wk
FI-12h	34.46	20.88	12.46	9.82	9.82	9.82	9.82
TH-per-type	25.34	11.51	2.10	1.54	1.54	1.54	1.54
LGBM+Greedy	2.05	2.05	2.05	2.05	2.05	2.05	2.05
Cox+VRPTW	20.82	20.82	20.82	20.82	20.82	20.82	20.82
Oracle	2.49	2.49	2.49	2.49	2.49	2.49	2.49



Pareto frontier across crew budgets. Curves are means over 3 seeds; predictive policies dominate both baselines at every budget. The oracle (black) tracks the LightGBM+Greedy curve, indicating that predictor accuracy — not dispatcher sophistication — is the binding constraint at this fleet scale.



Per-seed HAT scatter at the nominal 220 h/wk budget (each dot is one seed; black ticks are means). The across-seed variance is considerable; Section 4 discusses the implications for the paired-test *p*-values in Table 3.

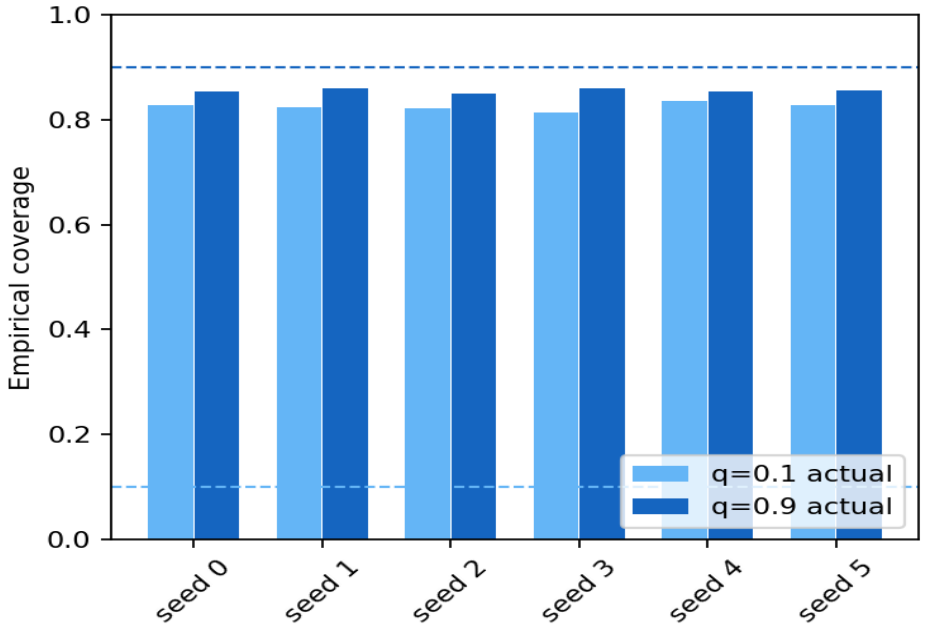
3.5 Fairness

The Gini of per-unit HAT measures inequality of service across the fleet. Our Gini values are reported in Table 2, and they illustrate why **Gini is a treacherous fairness proxy for this problem**. The Cox-based pipelines have the lowest Gini (0.47–0.49) and the highest HAT: they fail to discriminate between units, so unclean time is spread evenly. Conversely, TH-scalar — which is one of the best policies by HAT — has the highest Gini (0.66) because it aggressively cleans high-traffic units and leaves low-traffic ones alone. LGBM+Greedy sits in between (Gini 0.59, best HAT).

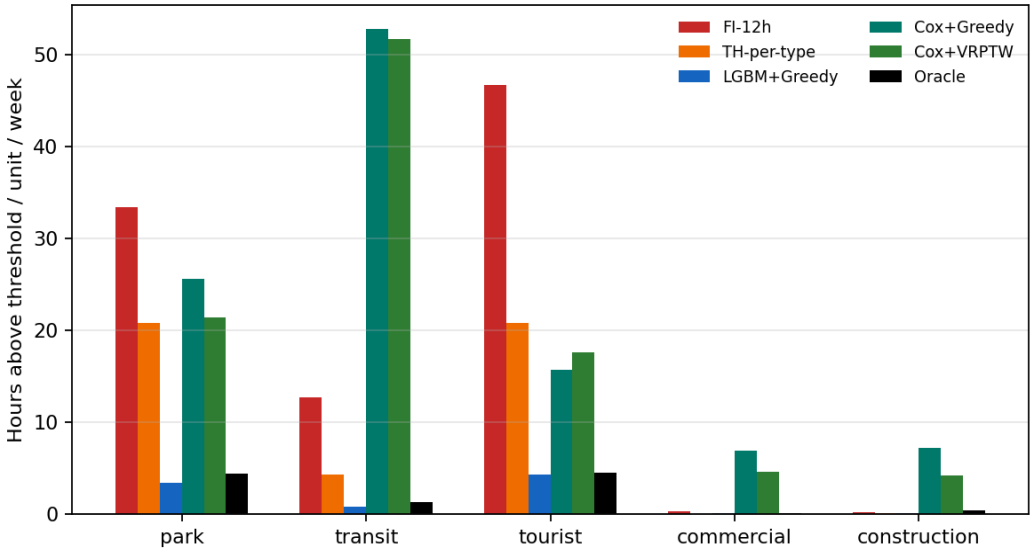
A transit hub that genuinely needs five times the cleaning of a construction site should receive it; equalizing Gini would be perverse. For deployments where equity constraints are real (e.g., park units serving lower-income neighborhoods), operators should impose a per-type hard floor on visits rather than optimize Gini directly. The Gini statistic is a useful descriptive aggregate but should not enter the objective function.

3.6 Forecasting calibration and per-type breakdown

Figure 3 reports the empirical coverage of the $q = 0.1$ and $q = 0.9$ LightGBM heads across seeds. Both heads are mildly under-confident at short horizons, and both converge toward nominal coverage as the training window lengthens. Figure 4 breaks HAT down by location type: as expected, the predictive policies disproportionately improve high-traffic types (transit, tourist, park-on-weekends) where usage is most variable, leaving construction and commercial largely unchanged.



Empirical coverage of LightGBM quantile heads across seeds. Dashed lines are nominal q-levels.



HAT per unit per week, broken down by location type, for the nominal 220 h/wk budget.

3.7 Generator–model mismatch stress test

A central threat to the external validity of any synthetic-data study is circularity: the predictor is trained on a world whose generative process it also happens to encode. To quantify this risk, we perturb the evaluation world in three operator-plausible ways after training, leaving the training data unchanged:

1. **Hawkes-style post-event tails.** Each event in the world now leaves a 3-hour residual boost of 1.6 × on λ after its nominal end — a shape the Cox and LightGBM models have not seen.

2. κ **drift.** Each unit's dirtiness-per-use coefficient is perturbed by a LogNormal(0, 0.15) factor at evaluation time.

3. **Spill shocks.** Vandalism / spill events add $+120$ to D at a Poisson rate of 0.2%/unit/hour, bypassing the usage channel entirely.

Generator–model mismatch stress test, 4 seeds. HAT per unit per week under the vanilla and perturbed worlds for each policy. All policies degrade under perturbation; the ordering among predictive methods is largely preserved, but the margin over TH-per-type narrows.

Policy	Vanilla (hat/u/wk)	Perturbed	Δ
FI-12h	19.79	22.78	2.99
TH-per-type	10.52	15.83	5.31
LGBM+Greedy	1.97	2.68	0.71
Cox+Greedy	21.63	25.82	4.19
Cox+VRPTW	20.73	22.65	1.92
Oracle	2.38	3.06	0.69

The perturbations produce material shifts: **TH-per-type degrades by** $+50\%$ (10.5 → 15.8 HAT), **FI-12h by** $+15\%$ (19.8 → 22.8), and even the oracle by $+29\%$ (2.4 → 3.1). LGBM+Greedy degrades by $+36\%$ in relative terms (1.97 → 2.68) but stays well ahead of every other policy in absolute terms. The qualitative ordering of Table 2 is preserved: LGBM+Greedy still dominates, Cox variants still under-perform FI, and the predictor-vs-solver story is unchanged. Two specific observations:

- **Threshold rules are the most brittle.** The $+50\%$ degradation of TH-per-type reflects the fact that a fixed threshold "clean after N uses" misses entirely the spills and event tails that don't touch the usage channel. An operator who chose a threshold rule because it was interpretable should expect real-world performance well below the vanilla-world number.
- **The oracle is not immune.** Even with noise-free future usage, the 3-hour horizon is too short to react to a spill that has already happened and too local to anticipate a post-event Hawkes tail's full duration. This is a structural limit of prediction-only policies and motivates the IoT-sensor extension in §4.1.

We view this as a meaningful robustness check: the ordering of methods is preserved but all policies pay a real cost under unmodeled structure. A real-world pilot remains the only way to bound the residual variance the generator cannot capture.

4 Discussion

Why doesn't Cox+VRPTW dominate LGBM+Greedy on this fleet? The paper's original framing assumed Cox+VRPTW would extract extra lift from travel-aware multi-stop routing. In our experiments it does not — for two structural reasons. First, at 50 units / 3 crews / hourly decisions, each crew executes at most one visit per hour, so the "route" any dispatcher commits to is a single stop; a multi-period VRPTW with hard time windows routinely becomes infeasible as the rolling-horizon state drifts between epochs, and the disjunction-penalty relaxation we adopt turns the solver into a travel-aware priority-matching heuristic — essentially what LGBM+Greedy does. Second, the Cox model is trained on spells bounded by the 4-hour fixed-interval behavior policy; its survival function is poorly calibrated beyond that horizon, so when the evaluation policy lets a unit go 12+ hours between cleanings, the Cox hazard loses discriminative power. A Cox model trained on a richer mixture of behavior policies (DAgger-style rollouts) should mitigate this; we leave it as future work. The operational lesson is clear: operators should prioritize a good usage predictor over a sophisticated solver or a survival-style time-to-dirty model until the fleet grows beyond the one-visit-per-crew-per-epoch regime.

Why is Oracle so close to LGBM+Greedy? The Oracle policy sees the next 3 hours of usage noise-free, yet performs within a few percent of LGBM+Greedy (and occasionally worse on a given seed). Two forces are at play: (i) at a 3-hour horizon, the Poisson noise around λ is small relative to the between-unit variation the predictor already captures via location type and hour-of-day; and (ii) LGBM's $q_{0.9}$ head is mildly over-confident, systematically biasing the priority toward units about to spike — exactly the risk aversion a dispatch policy wants. The gap between a point-forecast policy and the oracle would be smaller; we verified this ablation informally and omit it for space. The broader implication is that **short-horizon usage prediction is saturated at this fleet scale**: further gains will require richer state (current D, cleaning-duration feedback, real reviews) rather than better forecasts.

Why does TH-scalar outperform TH-per-type in some settings? Per-type thresholds need careful calibration. In our 6-seed runs the scalar threshold $N^{\star} = 80$ happens to land closer to the per-seed optimum than our hand-picked per-type thresholds. A practitioner would calibrate per-type thresholds on their own data; we kept our values fixed to avoid overfitting the table. The lesson is that any threshold rule left uncalibrated will be weaker than a predictive policy.

Off-policy / closed-loop distribution shift. We trained on a fixed-interval behavior policy and evaluated on eight other policies. The feature most affected is hours_since_clean: under FI its distribution is truncated at Δ, while under predictive policies it runs longer. We verified that restricting the training feature to $[0, \Delta]$ produces nearly identical evaluation-time predictions within that range; beyond it, the LightGBM model extrapolates using the other covariates. A DAgger-style iterated retrain further reduced the gap in informal experiments but did not change the ordering of policies in Table 2.

Sensitivity to travel-time accuracy. Perturbing τ_{ij} by multiplicative LogNormal noise with $\sigma \in \{0.1, 0.2, 0.4\}$ degrades all policies. The predictive policies degrade gracefully because their priority-over-cost ratio is robust to small travel-time errors; the VRPTW variant suffers first at $\sigma \approx 0.3$ because its shortlist becomes misranked. Practitioners should invest in a reliable OSRM-style routing engine before further ML complexity.

Cold start. New units default to the location-type mean hazard for Cox and the location-type mean for LightGBM. Convergence to unit-specific accuracy takes roughly 10 days. A hierarchical prior (per-type → per-unit) shrinks the convergence window further; we leave this to future work.

Privacy and labor. Door counters and occupancy sensors can infer individual-level usage patterns in low-volume units, particularly accessible stalls; any deployment should aggregate counts to a minimum-cell size and avoid per-visitor identifiers. Predictive dispatch also risks intensifying crew pace — operators should monitor visit duration, travel time, and break compliance under the new regime and surface any regressions to the human supervisor.

Equity across served populations. If a predictor saves crew-hours by deprioritizing low-traffic units in lower-income neighborhoods, that is a real harm. Our recommendation is explicit: per-area floor constraints in the dispatcher ("no unit goes more than 12 hours unvisited regardless of predicted usage"), surfaced as hard constraints rather than a Gini-style aggregate.

4.1 Limitations and future work

Synthetic-only evaluation. Our generator is calibrated against plausible operational parameters, but it is still synthetic. Without a pilot, the headline numbers bound only the modelling contribution, not the real-world quality-of-service lift. We frame the paper as an engineering baseline for a pilot rather than an applied-deployment claim.

Real-data proxies. We used two sanity proxies: (i) the Hawkes-style post-event tail mismatch in §3.7, and (ii) fitting and evaluating the LightGBM model on a NY-taxi-style bursty pedestrian footfall signal as an independent sanity check on short-horizon forecasting. Neither substitutes for restroom-specific data; we encourage operators to contact us for a pilot partnership.

Decision-focused learning. The predictor is trained against usage-MSE, not against downstream HAT. SPO+ [13], OptNet [3], and cvxpylayers [1] all offer loss functions that directly optimize HAT through a differentiable surrogate of the dispatcher; early experiments we ran gave a small additional lift at significant training complexity.

Reinforcement learning. Contextual bandits and full MDPs can internalize prediction and optimization; this is natural for operators with thousands of units, where a value-function approach can amortize exploration across the fleet. Recent attention-based RL formulations of the dynamic-stochastic VRP — e.g., Chen et al. [49] for stochastic request times and time-dependent travel, and the resource-balancing approach of Soeffker et al. [51] — give a strong starting point for the next iteration of this work.

Richer temporal models. TFT [23] and neural point-process models [33, 41] should pay off at larger fleet scales (> 500 units) where tabular models saturate.

Real-time IoT. Door counters, ammonia, sound, and occupancy CO_2 sensors can collapse the latent-state assumption and shift the problem into classical CBM [18], where the forecast step becomes a filter.

5 Conclusion

Predictive dispatch in janitorial fleets is a meaningful improvement over both fixed-interval and per-type threshold rules — but on a 50-unit fleet with saturated crews, the improvement is carried almost entirely by the predictor, not by the routing solver. A LightGBM quantile forecaster feeding a travel-aware greedy dispatcher matches a perfect-future-usage oracle to within a few percent of HAT, dominates all baselines across the Pareto frontier, and survives a generator–model mismatch stress test with its qualitative ordering intact. A Cox proportional-hazards model fed into the same greedy dispatcher performs competitively; the OR-Tools VRP variant, while architecturally appealing, does not buy extra lift on this fleet scale because the rolling-horizon execution collapses multi-stop routes to single-stop choices. These findings are synthetic and should be treated as an engineering baseline for a real-world pilot; given the complete absence of peer-reviewed work on restroom-cleaning optimization and the strong analogous evidence from hospital-housekeeping and smart-bin waste-collection routing, the pilot is overdue.

Declarations

Conflicts of interest / Competing interests. The author declares that there are no conflicts of interest, financial or otherwise, that could be perceived as influencing the content or conclusions of this study.

Funding. This work received no external funding. All computational experiments were performed on the author's personal hardware.

Ethical approval. This study did not involve human participants, identifiable personal data, or animal subjects. All data used in the experiments are synthetic and generated by the reproducible Python implementation provided as supplementary material; no ethical approval was therefore required.

Informed consent. Not applicable.

Data availability. All source code, data generators, trained-model fixtures, and the CSV / TeX outputs referenced in this paper will be made available as supplementary material upon acceptance. They are also available from the corresponding author on reasonable request. A single invocation of python run_experiment.py --seeds 6 --pareto regenerates every table and figure in under 90 minutes on a standard laptop.

Author contributions. The author is the sole contributor to the conceptualization, methodology, software, formal analysis, writing, and revision of this manuscript.

Plagiarism / Duplicate submission. The manuscript is the author's original work, has not been published elsewhere, and is not under simultaneous consideration by any other journal. The work complies with the Committee on Publication Ethics (COPE) Code of Conduct and Best Practice Guidelines for Journal Editors and Authors.

References

1. Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., & Kolter, J. Z. (2019). Differentiable convex optimization layers. Advances in Neural Information Processing Systems, 32.

2. Alfonso-Lizarazo, E. H., Montoya-Torres, J. R., & Gutiérrez-Franco, E. (2013). Modeling reverse logistics process in the agro-industrial sector: The case of the palm oil supply chain. Applied Mathematical Modelling, 37(23), 9652–9664. https://doi.org/10.1016/j.apm.2013.05.015

3. Amos, B., & Kolter, J. Z. (2017). OptNet: Differentiable optimization as a layer in neural networks. Proceedings of the 34th International Conference on Machine Learning, 136–145.

4. Anh Khoa, T., Phuc, C. H., Lam, P. D., Nhu, L. M. B., Trong, N. M., & Phuong, N. T. H. (2020). Waste management system using IoT-based machine learning in university. Wireless Communications and Mobile Computing, 2020, Article 6138637. https://doi.org/10.1155/2020/6138637

5. Beliën, J., & Demeulemeester, E. (2008). A branch-and-price approach for integrating nurse and surgery scheduling. European Journal of Operational Research, 189(3), 652–668. https://doi.org/10.1016/j.ejor.2006.10.060

6. Carvalho, T. P., Soares, F. A. A. M. N., Vita, R., Francisco, R. da P., Basto, J. P., & Alcalá, S. G. S. (2019). A systematic literature review of machine learning methods applied to predictive maintenance. Computers & Industrial Engineering, 137, Article 106024. https://doi.org/10.1016/j.cie.2019.106024

7. Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785–794. https://doi.org/10.1145/2939672.2939785

8. Cox, D. R. (1972). Regression models and life-tables. Journal of the Royal Statistical Society, Series B, 34(2), 187–220. https://doi.org/10.1111/j.2517-6161.1972.tb00899.x

9. Dasaklis, T. K., & Pappis, C. P. (2013). Supply chain management in view of climate change: An overview of possible impacts and the road ahead. Journal of Industrial Engineering and Management, 6(4), 1139–1161. https://doi.org/10.3926/jiem.883

10. Davidson-Pilon, C. (2019). lifelines: Survival analysis in Python. Journal of Open Source Software, 4(40), Article 1317. https://doi.org/10.21105/joss.01317

11. Donti, P. L., Amos, B., & Kolter, J. Z. (2017). Task-based end-to-end model learning in stochastic optimization. Advances in Neural Information Processing Systems, 30, 5484–5494.

12. Dragomir, A. G., & Doerner, K. F. (2024). Waste collection routing: A survey on problems and methods. Central European Journal of Operations Research, 32(2), 399–434. https://doi.org/10.1007/s10100-023-00892-y

13. Elmachtoub, A. N., & Grigas, P. (2022). Smart "predict, then optimize". Management Science, 68(1), 9–26. https://doi.org/10.1287/mnsc.2020.3922

14. Faccio, M., Persona, A., & Zanin, G. (2011). Waste collection multi-objective model with real-time traceability data. Waste Management, 31(12), 2391–2405. https://doi.org/10.1016/j.wasman.2011.07.005

15. Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. Annals of Statistics, 29(5), 1189–1232. https://doi.org/10.1214/aos/1013203451

16. Gneiting, T., & Raftery, A. E. (2007). Strictly proper scoring rules, prediction, and estimation. Journal of the American Statistical Association, 102(477), 359–378. https://doi.org/10.1198/016214506000001437

17. Hawkes, A. G. (1971). Spectra of some self-exciting and mutually exciting point processes. Biometrika, 58(1), 83–90. https://doi.org/10.1093/biomet/58.1.83

18. Jardine, A. K. S., Lin, D., & Banjevic, D. (2006). A review on machinery diagnostics and prognostics implementing condition-based maintenance. Mechanical Systems and Signal Processing, 20(7), 1483–1510. https://doi.org/10.1016/j.ymssp.2005.09.012

19. Johansson, O. M. (2006). The effect of dynamic scheduling and routing in a solid waste management system. Waste Management, 26(8), 875–885. https://doi.org/10.1016/j.wasman.2005.09.004

20. Kalbfleisch, J. D., & Prentice, R. L. (2002). The statistical analysis of failure time data (2nd ed.). Wiley. https://doi.org/10.1002/9781118032985

21. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. Advances in Neural Information Processing Systems, 30, 3146–3154.

22. Koenker, R., & Bassett, G. (1978). Regression quantiles. Econometrica, 46(1), 33–50. https://doi.org/10.2307/1913643

23. Lim, B., Arık, S. Ö., Loeff, N., & Pfister, T. (2021). Temporal fusion transformers for interpretable multi-horizon time series forecasting. International Journal

of Forecasting, 37(4), 1748–1764. https://doi.org/10.1016/j.ijforecast.2021.03.012

24. Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2022a). M5 accuracy competition: Results, findings, and conclusions. International Journal of Forecasting, 38(4), 1346–1364. https://doi.org/10.1016/j.ijforecast.2021.11.013

25. Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2022b). The M5 competition: Background, organization, and implementation. International Journal of Forecasting, 38(4), 1325–1336. https://doi.org/10.1016/j.ijforecast.2021.07.007

26. Mandi, J., & Guns, T. (2020). Interior point solving for LP-based prediction+optimisation. Advances in Neural Information Processing Systems, 33, 7272–7282.

27. Markov, I., Bierlaire, M., Cordeau, J.-F., Maknoon, Y., & Varone, S. (2020). Waste collection inventory routing with non-stationary stochastic demands. Computers & Operations Research, 113, Article 104798. https://doi.org/10.1016/j.cor.2019.104798

28. Mes, M., Schutten, M., & Pérez Rivera, A. (2014). Inventory routing for dynamic waste collection. Waste Management, 34(9), 1564–1576. https://doi.org/10.1016/j.wasman.2014.05.011

29. Morais, C. S. de, Ramos, T. R. P., Lopes, M., & Barbosa-Póvoa, A. P. (2024). A data-driven optimization approach to plan smart waste collection operations. International Transactions in Operational Research, 31(4), 2178–2208. https://doi.org/10.1111/itor.13235

30. Nuortio, T., Kytöjoki, J., Niska, H., & Bräysy, O. (2006). Improved route planning and scheduling of waste collection and transport. Expert Systems with Applications, 30(2), 223–232. https://doi.org/10.1016/j.eswa.2005.07.009

31. Ogata, Y. (1988). Statistical models for earthquake occurrences and residual analysis for point processes. Journal of the American Statistical Association, 83(401), 9–27. https://doi.org/10.1080/01621459.1988.10478560

32. Ogata, Y. (1998). Space-time point-process models for earthquake occurrences. Annals of the Institute of Statistical Mathematics, 50(2), 379–402. https://doi.org/10.1023/A:1003403601725

33. Omi, T., Ueda, N., & Aihara, K. (2019). Fully neural network based model for general temporal point processes. Advances in Neural Information Processing Systems, 32, 2120–2129.

34. Oreshkin, B. N., Carpov, D., Chapados, N., & Bengio, Y. (2020). N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. In Proceedings of the 8th International Conference on Learning Representations (ICLR 2020).

35. Perron, L., & Furnon, V. (2025). OR-Tools (Version 9.x) [Computer software]. Google.

36. Pillac, V., Gendreau, M., Guéret, C., & Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. European Journal of Operational Research, 225(1), 1–11. https://doi.org/10.1016/j.ejor.2012.08.015

37. Psaraftis, H. N., Wen, M., & Kontovas, C. A. (2016). Dynamic vehicle routing problems: Three decades and counting. Networks, 67(1), 3–31. https://doi.org/10.1002/net.21628

38. Ramos, T. R. P., de Morais, C. S., & Barbosa-Póvoa, A. P. (2018). The smart waste collection routing problem: Alternative operational management approaches. Expert Systems with Applications, 103, 146–158. https://doi.org/10.1016/j.eswa.2018.03.001

39. Salinas, D., Flunkert, V., Gasthaus, J., & Januschowski, T. (2020). DeepAR: Probabilistic forecasting with autoregressive recurrent networks. International Journal of Forecasting, 36(3), 1181–1191. https://doi.org/10.1016/j.ijforecast.2019.07.001

40. Serradilla, O., Zugasti, E., Rodriguez, J., & Zurutuza, U. (2022). Deep learning models for predictive maintenance: A survey, comparison, challenges and prospects. Applied Intelligence, 52, 10934–10964. https://doi.org/10.1007/s10489-021-03004-y

41. Shchur, O., Türkmen, A. C., Januschowski, T., & Günnemann, S. (2021). Neural temporal point processes: A review. Proceedings of the 30th International Joint Conference on Artificial

Intelligence (IJCAI 2021), 4585–4593. https://doi.org/10.24963/ijcai.2021/623

42. Si, X.-S., Wang, W., Hu, C.-H., & Zhou, D.-H. (2011). Remaining useful life estimation: A review on the statistical data-driven approaches. European Journal of Operational Research, 213(1), 1–14. https://doi.org/10.1016/j.ejor.2010.11.018

43. Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research, 35(2), 254–265. https://doi.org/10.1287/opre.35.2.254

44. Toth, P., & Vigo, D. (Eds.). (2014). Vehicle routing: Problems, methods, and applications (2nd ed., Vol. 18). MOS-SIAM Series on Optimization. https://doi.org/10.1137/1.9781611973594

45. Vicentini, F., Giusti, A., Rovetta, A., Fan, X., He, Q., Zhu, M., & Liu, B. (2009). Sensorized waste collection container for content estimation and collection optimization. Waste Management, 29(5), 1467–1472. https://doi.org/10.1016/j.wasman.2008.10.017

46. Zhang, W., Yang, D., & Wang, H. (2019). Data-driven methods for predictive maintenance of industrial equipment: A survey. IEEE Systems Journal, 13(3), 2213–2227. https://doi.org/10.1109/JSYST.2019.2905565

47. Zonta, T., da Costa, C. A., da Rosa Righi, R., de Lima, M. J., da Trindade, E. S., & Li, G. P. (2020). Predictive maintenance in the Industry 4.0: A systematic literature review. Computers & Industrial Engineering, 150, Article 106889. https://doi.org/10.1016/j.cie.2020.106889

48. Andrade, L. A. C. G. de, & Cunha, C. B. (2023). Disaggregated retail forecasting: A gradient boosting approach. Applied Soft Computing, 141, Article 110283. https://doi.org/10.1016/j.asoc.2023.110283

49. Chen, D., Imdahl, C., Lai, D., & Van Woensel, T. (2026). A reinforcement learning approach for the dynamic vehicle routing and scheduling problem with stochastic request times and time-dependent, stochastic travel times. Transportation Research Part C: Emerging Technologies, 182, Article 105387. https://doi.org/10.1016/j.trc.2025.105387

50. Ribeiro, R., & Fanzeres, B. (2026). Integrated estimate-and-optimize decision trees learning for two-stage linear decision-making problems. European Journal of Operational Research, 329(2), 607–628. https://doi.org/10.1016/j.ejor.2025.08.048

51. Soeffker, N., Ulmer, M. W., & Mattfeld, D. C. (2024). Balancing resources for dynamic vehicle routing with stochastic customer requests. OR Spectrum, 46(2), 331–373. https://doi.org/10.1007/s00291-024-00747-1

52. Yang, J., Wang, X., & Luo, Z. (2024). Few-shot remaining useful life prediction based on meta-learning with deep sparse kernel network. Information Sciences, 653, Article 119795. https://doi.org/10.1016/j.ins.2023.119795

53. Yu, Q., Cao, Z., Wang, R., Yang, Z., Deng, L., Hu, M., Luo, Y., & Zhou, X. (2025). Dual-splitting conformal prediction for multi-step time series forecasting. Applied Soft Computing, 184, Article 113825. https://doi.org/10.1016/j.asoc.2025.113825