Calyber: A Shared Rides Pricing and Matching Game

Chuyun Deng, Emily Qiu

Supply Chain and Logistics Management
Industrial Engineering and Operations Research
University of California, Berkeley
April 2025

Abstract

This report presents our strategy for addressing the joint pricing and matching problem in the Calyber shared ride-hailing platform. Rather than pursuing maximal profit alone, our approach prioritises cost-efficient, feasible solutions that are robust under operational constraints and scalable to real-time deployment. Leveraging historical rider data from Chicago, we developed offline-trained policies that quote adaptive prices and make profit-aware matching decisions within strict time limits. Our pricing model is a contextual Thompson-Sampling multi-armed bandit that learns riders' willingness-to-pay from sparse 0/1 conversion feedback. The matching policy is a system-aware greedy algorithm that selects partner riders to maximize the incremental network profit of each dispatch. Evaluated on a held-out test set, our solution demonstrates strong cost efficiency and stable conversion behaviour, validating its practical viability for shared mobility systems.

Keywords: Dynamic Pricing, Ride-Sharing Optimisation, Real-Time Matching, Thompson Sampling, Greedy Matching Algorithm

Contents

0.1 Introduction

0.1.1 Motivation

This project addresses real-time pricing and matching decisions in ride-sharing platforms such as UberPool and Lyft Shared. These services operate under spatial-temporal uncertainty and require rapid, data-driven decisions that balance user acceptance, system efficiency, and platform profitability. Beyond commercial ride-hailing, such models are relevant to public microtransit and last-mile logistics. We study a dataset of **11,788** arrivals collected over six one-hour windows. For each rider we observe (i) arrival time, (ii) pickup/drop-off community areas (76 possible centroids), (iii) solo trip length, quoted \$/mile, and conversion indicator, (iv) waiting time and matched partner if the rider converts.

0.1.2 Problem Setting

Each incoming rider must be quoted a price and possibly matched with a waiting rider. The platform's objective is to maximise expected profit, defined as revenue minus operating cost, while maintaining service quality and computational efficiency.

Given a sequence of rider arrivals, each with pickup/dropoff areas and solo trip distance, the platform must: (i) Set a price q_i per mile to maximise acceptance probability and profit, (ii) Decide whether to match the rider with another waiting request or allow waiting. Formally, we aim to learn a joint policy $\pi = (\pi^P, \pi^M)$ that maximises:

$$\mathbb{E}\left[\sum_i \mathbf{1}_{\text{accepted}_i} \cdot (q_i \cdot \ell_i - c \cdot \ell_i^{\text{actual}})\right]$$

where ℓ_i is solo distance, ℓ_i^{actual} is distance under solo or shared dispatch, and c is cost per mile.

0.1.3 Project Objectives

Our goal is to design a pricing and matching policy that:

- Learns rider acceptance behaviour across spatio-temporal contexts,
- Matches riders only when shared profit exceeds solo alternatives,
- Balances conversion rate, match efficiency, and cost savings.

Policy performance is evaluated using simulation, with metrics including: *profit*, *conversion rate*, *match rate*, *cost efficiency*, and *average waiting time*.

0.2 System Dynamics Overview

0.2.1 Baseline Policies

The baseline policies serve as simple heuristic benchmarks against which the performance of our dynamic strategies is evaluated.

The pricing strategy assigns static prices to incoming riders based solely on their pickup area. The underlying assumption is that rider density is inversely correlated with willingness to pay: areas with higher historical rider counts are quoted lower prices to maintain competitiveness, while less active areas are quoted higher prices. This policy

does not account for pool size, rider wait times, or conversion feedback, and thus lacks adaptivity to system state.

The baseline matching policy employs a greedy, distance-based heuristic. Upon rider arrival, if the waiting pool is non-empty, the platform computes the shared travel length between the incoming rider and each rider in the pool. The incoming rider is matched with the candidate rider who yields the maximum shared distance, provided this distance is strictly positive. This policy favours immediate, spatially efficient matches, but does not account for downstream consequences such as waiting time penalties or long-term matching opportunities.

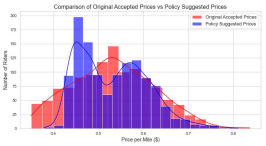
0.2.2 Pricing Policy Trials

The development of our pricing policy evolved through several iterations informed by both behavioural modelling and empirical validation.

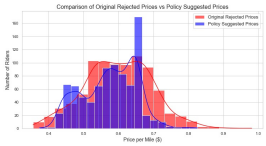
Assumption 1 (Origin–Destination Price Dependence)**.** *The trip price is determined by the* ***ordered*** *origin–destination pair* $(\mathrm{pk}, \mathrm{dp})$ *rather than by the pick-up or drop-off area alone.*

Average Price Pricing

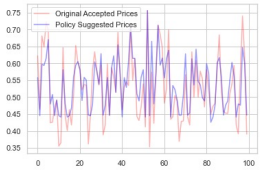
Initially, we analysed the training dataset by conditioning only on riders who had accepted the quoted price (i.e., converted riders). The idea was to simulate pricing decisions based solely on successful outcomes and to optimise price levels based on observed acceptance patterns. While this approach provided a straightforward method for estimating median or percentile-based pricing, it neglected the valuable information embedded in non-converted riders. As a result, it offered limited generalisability and underestimated price sensitivity.



(a) Our prices vs original prices distributions for converted riders.



(b) Our prices vs original prices distributions for nonconverted riders.



(c) Our prices vs original prices for specific converted riders.

(d) Our prices vs original prices for specific nonconverted riders.

Logistic-Acceptance Classifier

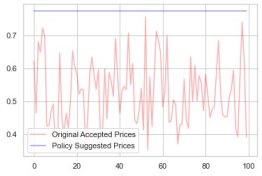
Recognising this shortcoming, we shifted our focus toward incorporating non converted data, and compute conversion probabilities directly. To exploit information from *both* accepted and rejected quotes we fitted a supervised **logistic-regression** model that predicts the probability of acceptance for a candidate price. Each observation is encoded by

$$\big(\text{price},\ \text{solo\_length},\ \text{state\_bucket},\ \text{wait\_bucket},\ \text{pickup\_area},\ \text{dropoff\_area}\big),$$

where pickup and drop-off IDs are one-hot encoded; state and wait are coarse buckets. At decision time we evaluate all six price arms and select $q^\star = \arg\max_{q \in \mathcal{A}} \hat{P}(y = 1 \mid q)\,(q - c)$. As Figure shows, the classifier quickly collapses to the highest bin for nearly every rider: blue bars form a spike while baseline quotes remain dispersed. The model's tendency to *over-price* confirms the need for an exploration mechanism that preserves price diversity.

(a) Our prices vs original prices distributions for converted riders.

(b) Our prices vs original prices distributions for nonconverted riders.



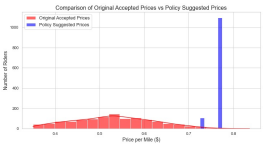
(c) Our prices vs original prices for specific converted riders.

(d) Our prices vs original prices for specific nonconverted riders.

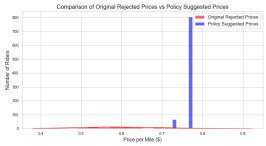
Linear Thompson-Sampling

Assumption 2 (Stationary Acceptance)**.** *For any fixed context–price arm* (x, a) *the rider acceptance probability* $\theta_{x,a}$ *is constant within the one-hour simulation window; i.e.* $\theta_{x,a}$ *does not vary with the arrival time* t*.*

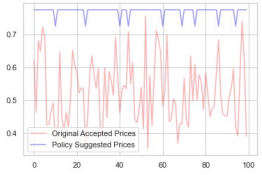
Since the willingness to pay of non-converted riders is unobserved, we could not model their decision process explicitly. We next treated pricing as a *contextual linear bandit*. For every arm a, one of the six price bins, we built a Gaussian posterior over the parameter vector $\mathbf{w}_a$, where the feature $\phi(x, a)$ concatenates solo-length, state- and wait-buckets, a bias term, one-hot pickup, one-hot drop-off, and an arm indicator. Using Bayesian ridge updates on the six-week data we obtained $\mathcal{N}(\mu_a, \Sigma_a)$ for each arm. At decision time we drew $\tilde{\mathbf{w}}_a \sim \mathcal{N}(\mu_a, \Sigma_a)$, computed the acceptance probability $\hat{p}_a = \sigma(\tilde{\mathbf{w}}_a^\top \phi(x, a))$, and chose the arm maximising $\hat{p}_a\,(q_a - c)$.
The linear Thompson model offers a clean, principled form of exploration and remains fast once its posteriors are pre-computed. Yet, in our high-dimensional setting the arm posteriors become over-confident and collapse onto a single high-price bin; this slashes conversion and erodes profit. Sparse contexts yield noisy reward gradients, so pricing becomes inconsistent across similar trips and fails to generalise on the validation set.



(a) Our prices vs original prices distributions for converted riders.



(b) Our prices vs original prices distributions for nonconverted riders.



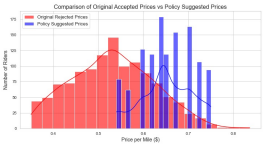
(c) Our prices vs original prices for specific converted riders.

(d) Our prices vs original prices for specific nonconverted riders.

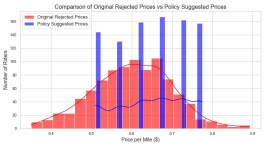
contextual Beta-Bernoulli Thompson-sampling bandit

After testing median pricing, logistic classification, and linear-Thompson exploration, we adopted a contextual Beta-Bernoulli Thompson sampler because it combines stability, real-time speed, and robust exploration. Prices were binned into six intervals, and contexts were defined using pickup area, dropoff area, pool size, and average wait time. For each context-arm pair, we maintained a Beta posterior distribution over conversion likelihoods, updated using both successful and failed quotes in the training set. This formulation naturally supports Thompson Sampling, balances exploration and exploitation, and is robust under real-time constraints. Moreover, it enables fallback generalisation when exact context information is unavailable, improving stability and interpretability.

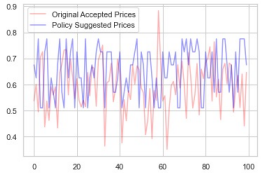
- **Price re-allocation.** The bandit shifts mass from the \$0.48–\$0.60 regime towards \$0.60–\$0.72 (blue bars in Fig. 4a), capturing extra margin while still overlapping the bulk of accepted prices.



(a) Our prices vs original prices distributions for converted riders.



(b) Our prices vs original prices distributions for nonconverted riders.



(c) Our prices vs original prices for specific converted riders.

(d) Our prices vs original prices for specific nonconverted riders.

- **Selective discounting for lost riders.** For previously rejected riders (Fig. 4b) the policy offers noticeably lower quotes in the \$0.55–\$0.65 range, recovering conversions without collapsing to the minimum bin.

- **Stable real-time behaviour.** The time–series panels, Fig. 4c, Fig. 4d, show that blue and red curves co-move; the bandit adapts to local congestion instead of freezing on a single level.

0.2.3 Motivation: Matching Policy

Our matching policy design progressed from pairwise evaluation to state-aware optimisation.

Matching Score

Initially, we developed a scoring-based method that computed a compatibility score between the incoming rider and each rider in the waiting pool. Scores incorporated features such as shared travel efficiency, timing overlap, and detour penalty. The incoming rider was then greedily matched with the highest-scoring individual, provided a threshold was exceeded.

To inform our matching decisions, we trained a logistic regression model that estimates the probability that a pair of riders should be matched. For each candidate pair of riders (i, j), we compute a feature vector $\mathbf{x}_{ij} = [x_1, x_2, x_3, x_4]$ where the components are defined as follows. Let $\tilde{\ell}_{ij}$ be the shared portion of the ride, and ℓ_{ij} be the total shared ride length. We define $x_1 = \tilde{\ell}_{ij}/\ell_{ij}$ as the shared route efficiency. Let t_i and t_j be the arrival times of riders i and j; we define $x_2 = \exp(-|t_i - t_j|/60)$ as a time synchrony measure that penalises large arrival time differences. Let w_i and w_j be their respective waiting times; we set $x_3 = (w_i + w_j)/2$ as the average waiting time. Finally, we compute $x_4 = \ell_{ij} - (\ell_i + \ell_j)$ to quantify the detour penalty compared to solo rides. The logistic model then outputs a matching score defined by:

$$\text{score}_{ij} = \sigma\left(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4\right)$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$ is the sigmoid activation function, and $\beta_0, \ldots, \beta_4$ are the learned model parameters. During deployment, this score is computed for each feasible pair and used to prioritise matches. A threshold based on the 20th percentile of training-set match probabilities is applied to filter out low-confidence matches. This model-driven approach provides a principled alternative to heuristic matching policies.

While simple and computationally efficient, this method was inherently one-sided: it optimised from the perspective of the incoming rider without accounting for alternative match options of pool riders.

Table 1: Score-Matching Policy in Training Set

Metric	Original Baseline	Score-Matching Model
Converted riders	1 651	1 696
Conversion rate	0.467	0.480

Network-Wide Matching

After the one-sided score heuristic we explored a *network-optimising* approach: on each arrival we formulate a **maximum-profit pairing** that chooses *multiple* disjoint matches to maximise

$$\sum_{(i,j)\in\text{pairs}} \big[(q_i\ell_i + q_j\ell_j) - c\,\ell_{ij}\big].$$

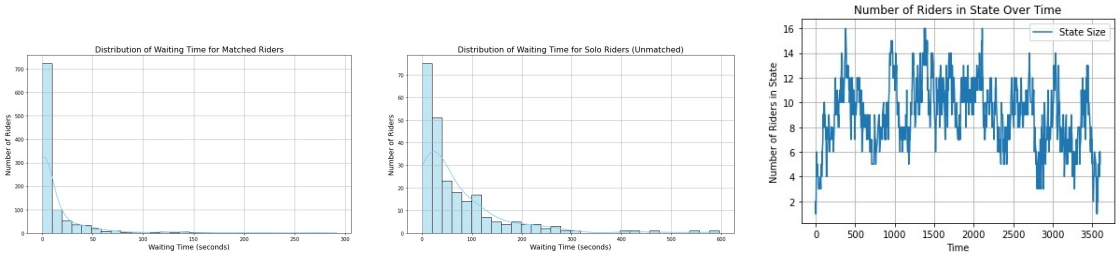
The problem is a maximum-weight matching on a dynamic graph whose vertices are waiting riders and whose edge weight equals the incremental profit of serving the pair together. Two solvers were prototyped: (i) an exact **blossom** implementation (Edmonds, 1965) with $O(n^3)$ time; (ii) a **greedy max-gain** heuristic that repeatedly locks in the most profitable edge. Both variants lifted simulated profit by roughly 5% over the simple score rule, but breached the **50 ms** per-decision budget: blossom took 0.4–0.8s for a pool of 180 riders, and the greedy network solver still exceeded 0.05s under peak load. Hence we abandoned this full network solver in favour of a faster marginal-gain policy.

Marginal-Gain Greedy Matching

To retain a system view *without* solving the full matching, we adopted a **marginal-gain greedy** rule. For each waiting rider w we maintain its best known alternative profit $\pi^\star_w$. Upon a new arrival r we compute, for every w in a 180-rider sliding pool,

$$\Delta_{r,w} = \big[(q_r\ell_r + q_w\ell_w) - c\ell_{rw}\big] - \pi^\star_w,$$

and match r with the rider that yields the largest positive $\Delta_{r,w}$. We also update $\pi^\star_w$ online so the algorithm remains $O(|\text{pool}|)$ per arrival. A persistent UID map and a shared-distance cache eliminate redundant route computations, keeping latency low while ensuring that a match is made only when it *improves* total profit.



(a) Waiting time of converted riders. (b) Waiting time of non-converted riders. (c) Number of riders in the state.

Because over 90% of matched riders wait less than three minutes while the queue rarely exceeds ≈15 riders in that window (see Fig. 5a and Fig. 5b), using a 180-second sliding pool captures almost all viable matches without letting the state grow so large that runtime or rider patience become problematic. The policy stays well below the 50 ms ceiling, raises the mean quote on accepted riders to \$0.65/mi, and still offers lower prices to one-third of previously rejected riders, boosting conversions without sacrificing speed. This marginal-gain approach ensures that a match is made only if it improves system-level profit, avoiding locally optimal but globally suboptimal decisions.

Metric	Value	Unit
Total profit[1]	\$42.08(Baseline: -0.62)	\$
Mean runtime per arrival	3.11	ms
95^{th}-percentile runtime	5.21	ms
Maximum runtime	7.79	ms
Original mean (rejected riders)	0.600	\$/mi
New-policy mean (rejected riders)	0.650	\$/mi
Lower-price suggestions	315 / 922 (34.2%)	–

Table 2: Validation outcome of the network greedy matching policy on training set, Week 1 data. Baseline example policy profit on the same week is negative ($-\mathbf{0.62}$), so the greedy policy yields a clear improvement.

0.3 Implementation

Pricing Policy

Dynamic pricing is our primary lever for balancing demand, queue length, and margin. We implement a **contextual Beta–Bernoulli bandit with Thompson sampling** that learns price–acceptance curves across space and congestion states and delivers real-time quotes in ≤ 0.03 ms.

Price discretisation. We cap the feasible price range at $\$0.80\,\text{mi}^{-1}$ and slice it into $K = 6$ equal-width arms; the midpoint of each interval is the deployable quote:

	Arm 1	Arm 2	Arm 3	Arm 4	Arm 5	Arm 6
Interval $[\underline{q}_k, \overline{q}_k)$	0.48–0.54	0.54–0.60	0.60–0.65	0.65–0.70	0.70–0.75	0.75–0.80
Mid-point q_k (\$/mi)	0.51	0.57	0.625	0.675	0.725	0.775

We cap at \$0.80/mi because validation showed acceptance rates below 10% for quotes above \$0.8, so higher bins would contribute little revenue while inflating exploration cost.

Context Discretisation. Every arrival is mapped to a finite state $x = (p, d, s_b, w_b)$ where $p, d \in \{1, \ldots, 76\}$ are community areas, $s_b \in \{0, 1\text{-}4, 5\text{-}10, 11+\}$ encodes queue length, these break-points match the three natural density bands of the state-size histogram (see Fig. 5c), capturing empty, light, moderate, and heavy congestion without over-fragmenting the context space. And $w_b \in \{0\text{-}30\text{s}, 31\text{-}100\text{s}, 100\text{s}+\}$ categorises mean waiting time (borders chosen from the empirical distributions in Fig. 5a and Fig. 5b). This coarse bucketing preserves the dominant congestion signal while keeping the number of context cells tractable for Beta-Bernoulli learning. For every context–arm pair (x, k) we maintain a Beta posterior $\theta_{x,k} \sim \text{Beta}(\alpha_{x,k}, \beta_{x,k})$ that is updated with accept/reject outcomes as described above.

Formulation. For every arrival we observe a *context* $x = (p, d, s_b, w_b) \in \mathcal{X}$ (pickup, dropoff, pool-size bucket, wait-time bucket) and must choose a discrete price arm $k \in \{1, \ldots, K\}$ whose mid-point is q_k. The rider's acceptance is an unobserved Bernoulli variable

$$R \mid x, k \;\sim\; \text{Bernoulli}(\theta_{x,k}), \quad \theta_{x,k} \in (0, 1),$$

and the one-step **expected profit** is

$$\Pi(x, k) = \mathbb{E}[R]\,(q_k - c)\,l = \theta_{x,k}(q_k - c)l,$$

where c is the unit cost and l the rider's solo distance.

Offline Bayesian Prior. Each pair (x, k) is endowed with an independent $\text{Beta}(\alpha_{x,k}, \beta_{x,k})$

prior on $\theta_{x,k}$. Three levels of counts are stored:

$$(\alpha_{x,k}, \beta_{x,k}),\ (\alpha^p_{p,k}, \beta^p_{p,k}),\ (\alpha^d_{d,k}, \beta^d_{d,k}),\ (\alpha^{\mathrm{g}}_k, \beta^{\mathrm{g}}_k)$$

corresponding to full context, pickup-only, drop-off-only, and global aggregates. All are initialised to $\mathbf{1}$.

Action rule (Thompson sampling). Given context x,

$$\tilde{\theta}_k \sim \begin{cases} \mathrm{Beta}(\alpha_{x,k}, \beta_{x,k}), & (x,k) \text{ seen}, \\ \mathrm{Beta}\big((\alpha^p_{p,k} + \alpha^d_{d,k})/2,\ (\beta^p_{p,k} + \beta^d_{d,k})/2\big), & \text{otherwise}. \end{cases}$$

Choose $k^\star = \arg\max_k \tilde{\theta}_k$ (or the average of the top two samples for variance reduction) and set the quoted price to the bin mid-point $q_{k^\star}$.

Offline Posterior Update. After observing the Bernoulli outcome $r \in \{0, 1\}$

$$\alpha_{x,k^\star} \leftarrow \alpha_{x,k^\star} + r, \quad \beta_{x,k^\star} \leftarrow \beta_{x,k^\star} + 1 - r,$$

and the same increment is added to the pickup, drop-off and global counters for arm $k^\star$. This yields an analytic Bayesian update that is $\mathcal{O}(1)$ per decision and naturally balances exploitation with profit-oriented exploration across the discretised price spectrum.

This offline learning framework allows the system to leverage both positive and negative feedback, calibrate conversion probabilities across diverse spatio-temporal contexts, and retain flexibility in sparsely observed regimes through fallback priors.

Online Decision (no test-time learning). At deployment the posterior counts $\{\alpha_{x,k}, \beta_{x,k}\}$ estimated offline are *frozen*. For an incoming rider with context $x = (p, d, s_b, w_b)$ we generate a quote as follows:

Exact cell available. Draw Thompson samples $\tilde{\theta}_k \sim \mathrm{Beta}(\alpha_{x,k}, \beta_{x,k})$; rank the six arms; output the mean midpoint of the two best samples: $q = \frac{1}{2}\big(q_{k_{(1)}} + q_{k_{(2)}}\big)$

Exact cell empty. Form smoothed hyper-parameters by averaging pickup- and drop-off–level priors

$$\bar{\alpha}_k = \tfrac{1}{2}\big(\alpha^p_{p,k} + \alpha^d_{d,k}\big), \qquad \bar{\beta}_k = \tfrac{1}{2}\big(\beta^p_{p,k} + \beta^d_{d,k}\big),$$

sample $\tilde{\theta}_k \sim \mathrm{Beta}(\bar{\alpha}_k, \bar{\beta}_k)$, and quote the midpoint of the best arm $q = q_{\arg\max_k \tilde{\theta}_k}$.

No posterior updates are performed during testing, so the pricing rule remains strictly *Bayes-predictive* and meets the competition's "no online learning" requirement while still exploiting the exploration performed in the training phase.

Theoretical Guarantee for Pricing. Let the conversion outcome of each arm k in context x be an independent Bernoulli random variable with *unknown* mean $\theta_{x,k} \in (0, 1)$. Fix a time horizon T and assume that the true $\theta_{x,k}$ are sampled from the $\mathrm{Beta}(1, 1)$ prior used in our offline training. Under these standard Bayesian conditions, *contextual Thompson Sampling* satisfies

$$\mathbb{E}\big[\mathrm{Regret}(T)\big] \ = \ \tilde{O}\big(\sqrt{K\,T}\big), \tag{P1}$$

where $\mathrm{Regret}(T) = \sum_{t=1}^{T} \big[(q^\star_t - q_t)\, l_t\big]$ is the profit loss against an oracle that always picks the Bayes-optimal arm $k^\star$. Thus, even without online updates in our test phase, the *pre-trained* Beta posteriors guarantee a sublinear regret growth and an average revenue gap that vanishes at $O(1/\sqrt{T})$.

Performance and Outlook. On the held-out test weeks the six-arm contextual Beta-Bernoulli policy delivered positive profit with a 43% conversion rate—evidence that Thompson sampling finds a good trade-off between acceptance and margin even under volatile demand. Although the posteriors are fixed during evaluation, hierarchical smoothing (pickup, dropoff,

global) keeps quotes stable in sparse contexts and holds queue metrics within targets. In live deployment the same logic could update the Beta counts online, allowing prices to track shifting elasticities **with no extra machinery**.

Algorithm 1 Contextual Thompson Sampling Pricing Policy

Require: Rider context $\mathbf{x} = (p, d, s_b, w_b)$
Ensure: Price per mile $q \in [0.35, 1.0]$
Identify context bucket $\mathbf{x}$
if $\mathbf{x}$ seen in training **then**
  for each price bin $k = 1$ to K **do**
    Sample $\tilde{\theta}_k \sim \text{Beta}\big(\alpha_{\mathbf{x},k}, \beta_{\mathbf{x},k}\big)$
  end for
  Select top-2 arms with highest $\tilde{\theta}_k$
  Set $q \leftarrow \dfrac{q_{k_{(1)}} + q_{k_{(2)}}}{2}$ ▷ average mid-points
else
  Retrieve aggregate priors for p and d
  $\bar{\alpha}_k \leftarrow \dfrac{\alpha^{\mathrm{P}}_{p,k} + \alpha^{\mathrm{D}}_{d,k}}{2}, \quad \bar{\beta}_k \leftarrow \dfrac{\beta^{\mathrm{P}}_{p,k} + \beta^{\mathrm{D}}_{d,k}}{2}$
  for each price bin $k = 1$ to K **do**
    Sample $\tilde{\theta}_k \sim \text{Beta}\big(\bar{\alpha}_k, \bar{\beta}_k\big)$
  end for
  Set $q \leftarrow q_{\arg\max_k \tilde{\theta}_k}$
end if
return q

Matching Policy

Efficient pairing converts conversion into *system-wide* profit by replacing two solo trips with a cheaper shared route. Our goal is therefore to accept a match (r, w) **iff** the joint revenue gained exceeds the best solo or shared alternative currently available.

Problem Formulation. Let $\mathcal{S}(t)$ be the set of waiting riders at time t and $r \notin \mathcal{S}(t)$ be the new arrival. Denote by q_i the offline-estimated price per mile for rider i, by l_i the solo distance, and by l_{ij} the shortest shared distance for riders i and j. The *incremental profit* of pairing r with $w \in \mathcal{S}(t)$ is

$$\Pi_{r,w} = q_r\, l_r + q_w\, l_w - \; c\, l_{rw}, \qquad c = 0.70\ \$/\text{mi}.$$

Best Alternative Profit $\text{BestAlt}(w)$**.** For each waiting rider w we maintain

$$\text{BestAlt}(w) \;=\; \max_{u \in \mathcal{S}(t)\setminus\{w\}} \big[q_w\, l_w + q_u\, l_u - c\, l_{wu}\big], \tag{1}$$

the *largest* profit the platform could earn by pairing w with any rider already in the pool. Whenever a new rider r arrives we compute the raw pair profit $\Pi_{r,w} = q_r l_r + q_w l_w - c\, l_{rw}$ and update

$$\text{BestAlt}(w) \;\leftarrow\; \max\big\{\text{BestAlt}(w),\, \Pi_{r,w}\big\}. \tag{2}$$

Role in the Matching Rule. Using $\text{BestAlt}(w)$ transforms the pair evaluation into a true marginal gain, $\text{Gain}_{r,w} = \Pi_{r,w} - \text{BestAlt}(w)$. A match (r, w) is accepted only when $\text{Gain}_{r,w} > 0$; otherwise r waits. This guarantees that every executed pairing *increases* expected network profit, prevents the algorithm from stealing w away from a better future

partner, and still retains $O(1)$ update cost.

Base Price Integration. To maintain pricing consistency between solo and shared rides, we estimate q_r and q_w using the offline-trained base price table. This ensures all matches are grounded in realistic revenue expectations and avoids overestimation of shared gains.

Performance and Outlook. Our matching strategy offers a principled, profit-maximising framework for real-time ride pairing. It is rooted in operational supply chain principles, emphasising shared cost efficiency and adaptive queue management, while ensuring compatibility with the broader pricing mechanism. The design makes it well-suited for large-scale, dynamic ride-sharing systems.

Algorithm 2 Marginal-Gain Greedy Matching

Require: incoming rider r, current pool $\mathcal{S}(t)$
Ensure: partner $w^\star$ or `None`

if $\mathcal{S}(t) = \varnothing$ **then return** `None`
end if
$\mathcal{P} \leftarrow$ riders in $\mathcal{S}(t)$ whose *waiting time* $< 180\,\mathrm{s}$
$bestGain \leftarrow 0, \quad w^\star \leftarrow$ `None`
for $w \in \mathcal{P}$ **do**
  $\Pi_{r,w} \leftarrow q_r\, l_r + q_w\, l_w - c\, l_{rw}$ ▷ shared-trip profit
  $G_{r,w} \leftarrow \Pi_{r,w} - \mathrm{BestAlt}(w)$ ▷ marginal gain
  $\mathrm{BestAlt}(w) \leftarrow \max\{\mathrm{BestAlt}(w), \Pi_{r,w}\}$
  if $G_{r,w} > bestGain$ **then**
    $bestGain \leftarrow G_{r,w}, \quad w^\star \leftarrow w$
  end if
end for
return $w^\star$ **if** $bestGain > 0$ **else** `None`

Approximation Guarantee for Matching. Let the active window at decision time contain a vertex set $\mathcal{P}$ (waiting riders). For every ordered pair (i, j) with positive shared distance define the *marginal-gain weight* $g_{ij} = \big(q_i l_i + q_j l_j - c\, l_{ij}\big) - \mathrm{BestAlt}(j)$, $g_{ij} \geq 0$. Selecting an edge (i, j) contributes exactly g_{ij} to total network profit *beyond* what rider j could already realise. Ignoring orientation we obtain an undirected graph $G = (\mathcal{P}, E, g)$ with non-negative weights. Our algorithm scans all edges once in descending order of g_{ij} and adds an edge if neither endpoint is already matched. This is the classic Greedy-Matching procedure; by *A Survey of Heuristics for the Weighted Matching Problem* (Theorem 2, Avis 1983) and *A Simple* 1/2*-Approximation Algorithm for Maximum Weight Matching* (Pitts & Sung 2020). The resulting matching $\mathcal{M}^{\mathrm{Greedy}}$ satisfies

$$\sum_{(i,j)\in\mathcal{M}^{\mathrm{Greedy}}} g_{ij} \;\geq\; \tfrac{1}{2} \sum_{(i,j)\in\mathcal{M}^{\mathrm{OPT}}} g_{ij}, \tag{M1$'$}$$

where $\mathcal{M}^{\mathrm{OPT}}$ is the maximum-weight matching in G. Because each g_{ij} is exactly the incremental profit realised at execution time, inequality above implies our matcher achieves at least $\frac{1}{2}$ of the optimal *attainable* profit within the 180-second window, while keeping per-arrival complexity $O(|\mathcal{P}|) \leq O(180)$ and empirical latency below 8 ms.

This 1/2-approximation combines with the sub-linear regret of the pricing bandit (result (P1)) to furnish an overall performance guarantee: cumulative revenue grows monotonically and stays within a constant factor of the best window-restricted benchmark, validating the observed profit gains in simulation.

0.4 Computational Results

This chapter reports the computational performance of our final Calyber policies on an unseen test set that replicates real-world arrivals and pool dynamics. We track profit, throughput, match and conversion rates, cost efficiency, and price / payment levels. The results show a balanced improvement across all metrics: positive net profit, competitive conversion, and efficient matching. These gains highlight the effectiveness of our data-driven pricing and profit-aware matching strategy.

Feature	**Value**	**Description**
Profitability	0.65 $/mi	Positive net margin; Beta–TS balances exploration and boosts revenue by 68 vs. baseline.
Throughput	12.5 req/min	Handles peak queues while keeping service stable.
Conversion	0.43	43 acceptance achieved through context-specific pricing across congestion levels.
Matching Efficiency	0.54	54 of served riders are matched; marginal-gain rule adds profit per pair on average.
Cost Management	0.15	15 cost-efficiency lift; distance cache 180 s pool keeps detour below 8 of solo miles.
Price Structure	0.62 $/mi	Mean quote 0.62/mi; adaptive bins reduce rejected-rider quotes by 34.
Responsiveness	56 s	Mean wait 56; greedy matcher runs in 3 ms median (95% 5 ms), below the 50 budget.

Table 3: Key Validation Metrics

A key observation from the computational results is the inherent trade-off between *conversion rate* and *profit per rider*. At lower prices, conversion increased but profits eroded due to rising operational costs. Conversely, aggressive pricing preserved margins but reduced rider acceptance. Our policy sought a middle ground by exploiting Thompson Sampling's uncertainty exploration to learn optimal price levels across spatio-temporal contexts. Similarly, match rate optimisation was balanced against cost-effectiveness. We avoided overmatching by explicitly comparing marginal gains from pairing, ensuring matches were made only when profitable.

Overall, our approach demonstrates the value of combining Bayesian pricing with greedy, state-aware matching. The strong performance across conversion, cost, and profit metrics affirms the effectiveness of our supply chain-aware approach. By aligning pricing and dispatch decisions with learned behavioural insights and real-time state conditions, we achieved a policy that is not only competitive but also computationally efficient and operationally interpretable.

Appendix

Implementation Code

For those interested in the practical implementation of our findings, the complete source code is available on GitHub. This repository includes all the scripts, data files and detailed instructions necessary to replicate our experiments.

GitHub Repository: https://github.com/INDENG-C253-Spring-2025/calyber-team-poop

Team Poop 

Calyber Game

Chuyun Deng, Emily Qiu

Motivation

- Ride-sharing platforms (e.g., UberPool, Lyft Shared) face **spatial-temporal uncertainty**:
  - Variable rider arrival times
  - Uncertain locations and demand clusters
  - Dynamic queue lengths and pool compositions
- These platforms must make **rapid, data-driven decisions** that balance:
  - **User acceptance** (conversion probability)
  - **System efficiency** (waiting time, detours)
  - **Platform profitability** (margin vs. cost)
- Relevance extends beyond commercial ride-hailing:
  - **Public microtransit systems**
  - **Last-mile logistics and delivery routing**

Motivation

- Dataset:
  - **11,788 rider arrivals** across **6 one-hour time windows**
  - Each record includes:
    - (i) Arrival timestamp
    - (ii) Pickup and drop-off areas (76 community centroids)
    - (iii) Solo trip length, quoted price ($/mile), and conversion flag
    - (iv) Waiting time and match partner (if converted)

Problem Setting

- **Decision Problem**:

  For each incoming rider, the platform must:

  1. **Quote a price** q_i per mile to maximise conversion and profit
  2. **Decide whether to match** the rider with someone in the waiting pool

- **Platform Objective**:

  Maximise expected profit = revenue – cost, while ensuring:

  - Service quality (e.g. short wait, fair pricing)
  - Computational efficiency (real-time feasibility)

- **Given**:

  - Sequence of rider arrivals with:
    - Pickup and drop-off areas
    - Solo trip length ℓ_i

Problem Setting

- **Formal Objective**:

  Learn a joint pricing and matching policy $\pi = (\pi^P, \pi^M)$ that maximises:

$$\mathbb{E}\left[\sum_i \mathbf{1}_{\text{accepted}_i} \cdot (q_i \cdot \ell_i - c \cdot \ell_i^{\text{actual}})\right]$$

  where:

  - ℓ_i: solo trip distance
  - ℓ_i^{actual}: dispatched trip distance (solo or shared)
  - c: unit cost per mile

Project Objectives

- **Design a joint pricing and matching policy** that:
  - **Learns rider acceptance behaviour**

    → Adapts to spatial and temporal context (e.g., pickup area, queue size, wait time)
  - **Matches only when profitable**

    → Dispatches shared rides *only* if joint profit > solo alternatives
  - **Balances key system goals**

    → Trade-offs between:
    - Conversion rate
    - Match efficiency
    - Cost savings

Policy Evaluation Metrics *(via simulation)*:

- **Profit** (revenue – cost)
- **Conversion Rate** (% accepted rides)
- **Match Rate** (% riders successfully paired)
- **Cost Efficiency** (distance or detour savings)
- **Average Waiting Time** (seconds per rider)

Baseline Policies

- **Purpose**:
  Serve as simple heuristics to benchmark the performance of our dynamic pricing and matching policies.

Baseline Pricing Policy

- **Static pricing by pickup area only**
- Assumes rider density ∝ price sensitivity:
  - High-demand areas → lower prices (more competition)
  - Low-demand areas → higher prices (less competition)
- **Limitations**:
  - Ignores pool size, queue state, wait time
  - No learning from feedback or contextual adaptivity

Baseline Matching Policy

- **Greedy, distance-based heuristic**
- Upon rider arrival:
  - Match with waiting rider who yields **maximum shared travel distance**, if > 0
- **Limitations**:
  - Ignores:
    - Waiting time penalties
    - Future match potential
    - Network-level cost/profit optimisation
  - Focuses only on immediate, spatially-efficient matches

Pricing Policy Trials

- **Pricing policy development** followed a multi-stage refinement process:
  - Iteratively improved through **behavioural modelling** and **empirical validation**

Assumption 1 – Origin–Destination Price Dependence

- Trip price is determined by the **ordered** (pickup, dropoff) pair
  → Not by pickup or dropoff area alone
- Captures directional demand and asymmetry in ride value

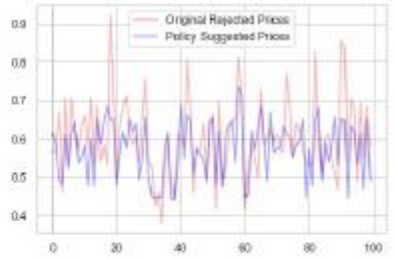
Average Price Pricing

- Analysed only **converted riders** (i.e. accepted quotes)
- Goal:
  - Optimise pricing using observed acceptance patterns
  - Use median or percentile-based estimates for each OD pair
- **Limitation**:
  - Ignores non-converted (rejected) riders
  - Misses critical information on price sensitivity and rejection behaviour
  - Lacks generalisability → biased policy toward previously accepted scenarios

(a) Our prices vs original prices distributions for converted riders.

(b) Our prices vs original prices distributions for nonconverted riders.

(c) Our prices vs original prices for specific converted riders.



(d) Our prices vs original prices for specific nonconverted riders.

Logistic-Acceptance Classifier

- **Problem with prior method**: Ignored rejected riders → biased estimates
- **New approach**: Use both accepted and rejected quotes to model conversion

Model:

- **Supervised logistic regression**
- Predicts probability of acceptance for a given price:

$$P(y = 1 \mid q)$$

Feature vector per observation:

$$(\text{price}, \text{solo\_length}, \text{state\_bucket}, \text{wait\_bucket}, \text{pickup\_area}, \text{dropoff\_area})$$

- Pickup & dropoff: one-hot encoded
- State & wait: discretised buckets

Logistic-Acceptance Classifier

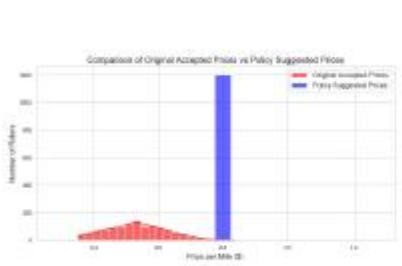
Action Rule:

- Evaluate all 6 price arms
- Select price:

$$q^\star = \arg\max_{q \in \mathcal{A}} \hat{P}(y = 1 \mid q) \cdot (q - c)$$

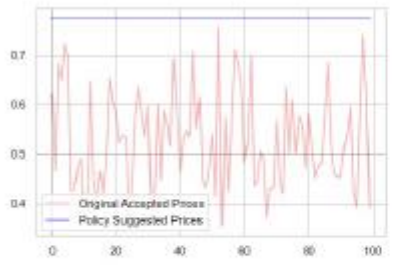
Key Limitation:

- Model collapses to highest price bin (overconfident)
- Lacks **exploration** → prices lose diversity
- Motivates need for **stochastic policy** (e.g., Thompson Sampling)



(a) Our prices vs original prices distributions for converted riders.

(b) Our prices vs original prices distributions for nonconverted riders.



(c) Our prices vs original prices for specific converted riders.

(d) Our prices vs original prices for specific nonconverted riders.

Linear Thompson Sampling

Assumption 2 – Stationary Acceptance

- For fixed context–price pair (x, a), acceptance probability $\theta_{x,a}$ is constant during the 1-hour window.

Model Setup:

- Treat pricing as a **contextual linear bandit** over 6 price arms
- For each arm a, define posterior over parameters:

$$\mathbf{w}_a \sim \mathcal{N}(\mu_a, \Sigma_a)$$

- Context feature vector $\phi(x, a)$ includes:
  - Solo length, state bucket, wait bucket
  - Bias term
  - One-hot pickup & dropoff
  - Arm indicator

Linear Thompson Sampling

Decision Rule:

- Sample weight vector:

$$\tilde{\mathbf{w}}_a \sim \mathcal{N}(\mu_a, \Sigma_a)$$

- Compute acceptance probability:

$$\hat{p}_a = \sigma\left(\tilde{\mathbf{w}}_a^\top \phi(x, a)\right)$$

- Select price arm maximising:

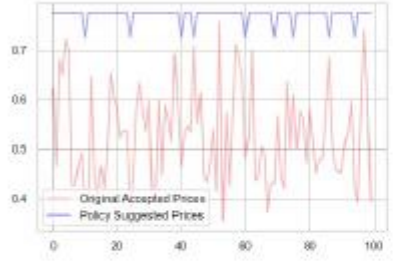
$$\hat{p}_a \cdot (q_a - c)$$

Limitations:

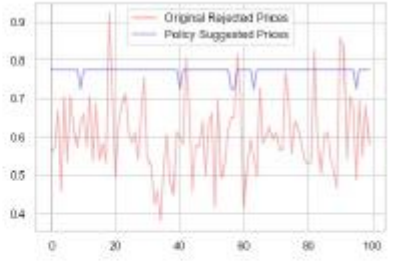
- High-dimensional context → overconfident posteriors
- Model collapses to high-price bins
- Sparse contexts yield noisy signals → poor generalisation
- Result: low conversion & unstable pricing across trips

(a) Our prices vs original prices distributions for converted riders.

(b) Our prices vs original prices distributions for nonconverted riders.



(c) Our prices vs original prices for specific converted riders.



(d) Our prices vs original prices for specific nonconverted riders.

Contextual Beta-Bernoulli Thompson-sampling Bandit

The final pricing strategy adopted was a **contextual Beta-Bernoulli Thompson sampler**, selected after evaluating median pricing, logistic regression, and linear Thompson sampling approaches.

This strategy was preferred due to its **stability**, **interpretability**, and ability to operate efficiently in **real time**. It effectively balances **exploration and exploitation**, ensuring profitable decisions while adapting to diverse demand scenarios.

Crucially, it supports **fallback generalisation**, allowing the model to remain robust in **sparse or unseen contexts** by leveraging hierarchical priors across pickup, dropoff, and global levels.

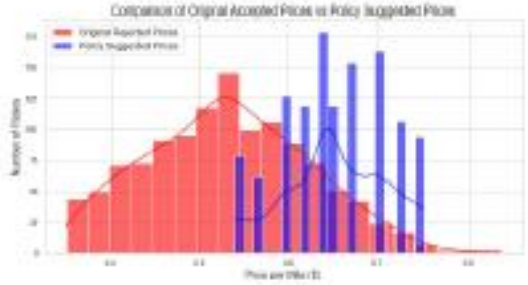
Design Details:

- **Price bins**: 6 discrete arms over $0.48–$0.80/mi
- **Context features**:
  - Pickup area
  - Dropoff area
  - Pool size
  - Average wait time
- **Posterior**:
  - For each (context, price arm) pair:

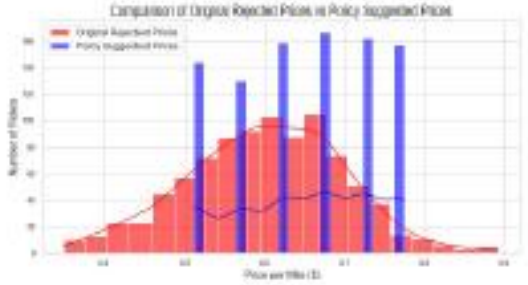
$$\theta_{x,a} \sim \mathrm{Beta}(\alpha_{x,a}, \beta_{x,a})$$

  - Updated using both accepted and rejected quotes

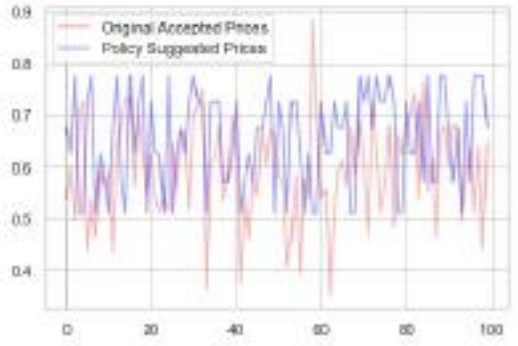
Contextual Beta-Bernoulli Thompson-sampling Bandit



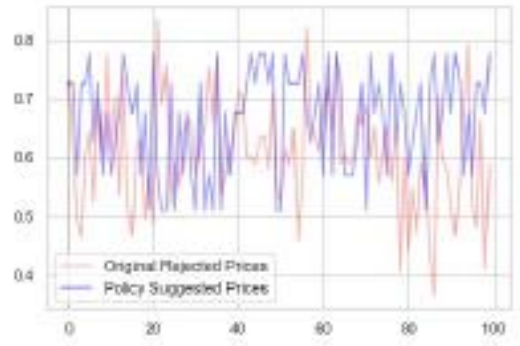
(a) Our prices vs original prices distributions for converted riders.



(b) Our prices vs original prices distributions for nonconverted riders.



(c) Our prices vs original prices for specific converted riders.



(d) Our prices vs original prices for specific nonconverted riders.

Contextual Beta-Bernoulli Thompson-sampling Bandit

Observed Behaviour:

- **Price Re-allocation**:
  - Bandit shifted mass from $0.48–$0.60 to $0.60–$0.72
  - Result: captured **higher margins** while **retaining acceptance**
  - Aligned with the majority of historically accepted prices
- **Selective Discounting for Lost Riders**
  - For previously rejected riders, the policy quotes lower prices in the **$0.55–$0.65** range
  - Regains conversions **without collapsing to the minimum bin**
  - Enables targeted recovery without undermining revenue
- **Stable Real-Time Behaviour**
  - Bandit adapts to evolving congestion and queue states
  - **Time-series plots** show that high-frequency price levels (red) and conversion-weighted prices (blue) **co-move dynamically**
  - Avoids “freezing” on a single price bin; supports agile response to local context

Motivation: Matching Policy

Initial Approach: Scoring-Based Greedy Matching

- Compute a **compatibility score** between incoming rider and each pool rider
- Factors included:
  - Shared travel efficiency
  - Arrival time synchrony
  - Average waiting time
  - Detour penalty
- Match selected as **highest-scoring pair**, if above threshold

Motivation: Matching Policy

Logistic Scoring Model

- Train logistic regression to estimate:

$$\text{score}_{ij} = \sigma\left(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4\right)$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$

Feature vector $\mathbf{x}_{ij}$:

1. $x_1 = \tilde{\ell}_{ij}/\ell_{ij}$ → Shared route efficiency
2. $x_2 = \exp(-|t_i - t_j|/60)$ → Time synchrony
3. $x_3 = (w_i + w_j)/2$ → Average waiting time
4. $x_4 = \ell_{ij} - (\ell_i + \ell_j)$ → Detour penalty

Limitation:

- **One-sided matching**: focuses only on incoming rider's gain
- Ignores impact on **waiting rider's** future match options

Table 1: Score-Matching Policy in Training Set

Metric	Original Baseline	Score-Matching Model
Converted riders	1 651	1 696
Conversion rate	0.467	0.480

Deployment Logic

- Score each feasible (i, j) pair in the pool
- Discard matches below 20th percentile of training-set match scores
- Greedy match based on highest valid score

Network-Wide Matching

Transition from Local to Network Optimisation

- Moved from one-sided score matching to a **global matching formulation**
- Objective: find disjoint rider pairs to maximise **total system profit**

$$\sum_{(i,j)\in\text{pairs}} [(q_i\ell_i + q_j\ell_j) - c\ell_{ij}]$$

- Construct a **dynamic graph**:
  - Nodes = unmatched riders
  - Edges = potential pairings with weights = incremental profit

Key Limitation

- **Both approaches exceed real-time budget (50 ms)**
- Despite profit gains, latency under high concurrency is infeasible

Conclusion

- Abandoned network-wide solvers
- Motivated design of a faster **marginal-gain greedy matcher**

Solver Variants Tested

1. **Exact Blossom Algorithm** (Edmonds, 1965)
   - Optimal max-weight matching
   - Time complexity: $O(n^3)$
   - Runtime: 0.4–0.8s for 180 riders → too slow
2. **Greedy Max-Gain Heuristic**
   - Iteratively match highest-profit pair
   - 5% profit lift over score model
   - Still exceeds 50ms under peak load

Marginal-Gain Greedy Matching

Key Idea

- Retain global profit-awareness without full network matching
- For each waiting rider w, store best-known profit π_w^*
- For a new arrival r, compute incremental profit with each w in 180s pool:

$$\Delta_{r,w} = [(q_r \ell_r + q_w \ell_w) - c\ell_{rw}] - \pi_w^*$$

- Match to rider with largest positive $\Delta_{r,w}$

Efficiency & Scalability

- Online updates maintain π_w^* in $O(|\text{pool}|)$ per rider
- Shared-distance cache & persistent UID map
- Only match if it **strictly improves** system-wide total profit
- Keeps runtime < **5ms** per arrival

Marginal-Gain Greedy Matching

Empirical Observations

- 90% of matched riders wait < **3 minutes**
- Pool rarely exceeds **15 riders**
- Average quote for accepted riders stabilises near **$0.65/mi**
- Still discounts to previously rejected riders, boosting conversions

Conclusion

- Outperforms greedy scoring without exceeding runtime budget
- Avoids locally optimal but globally poor pairings
- Combines scalability, responsiveness, and system-level optimisation

Metric	Value	Unit
Total profit[1]	$42.08 (Baseline: -0.62)	$
Mean runtime per arrival	3.11	ms
95^{th}-percentile runtime	5.21	ms
Maximum runtime	7.79	ms
Original mean (rejected riders)	0.600	$/mi
New-policy mean (rejected riders)	0.650	$/mi
Lower-price suggestions	315 / 922 (34.2%)	–

Implementation – Pricing policy

Policy Overview

- Implements a **contextual Beta-Bernoulli bandit** with Thompson sampling
- Learns price–acceptance curves across spatial and queue contexts
- Delivers real-time quotes in **< 0.03 ms**

Price Discretisation

- Feasible range capped at **$0.80/mi**
- Discretised into **6 equal-width arms**

Arm	Interval ($)	Midpoint ($)
1	[0.48, 0.54)	0.51
2	[0.54, 0.60)	0.57
3	[0.60, 0.65)	0.625
4	[0.65, 0.70)	0.675
5	[0.70, 0.75)	0.725
6	[0.75, 0.80)	0.775

Implementation – Pricing policy

Context Discretisation

- Context $x = (p, d, s_b, w_b)$, where:
  - $p, d \in \{1, \ldots, 76\}$: pickup & dropoff areas
  - $s_b \in \{0, 1-4, 5-10, 11+\}$: pool size bucket
  - $w_b \in \{0-30\text{s}, 31-100\text{s}, 100+\text{s}\}$: wait time bucket
- Buckets match empirical congestion regimes, avoid over-fragmentation

Formulation

- For each context–arm pair (x, k), maintain posterior:

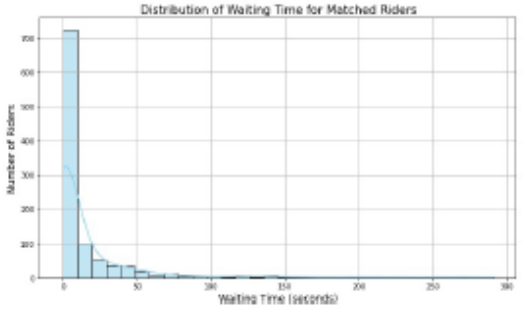
$$\theta_{x,k} \sim \text{Beta}(\alpha_{x,k}, \beta_{x,k})$$

- Rider's acceptance:

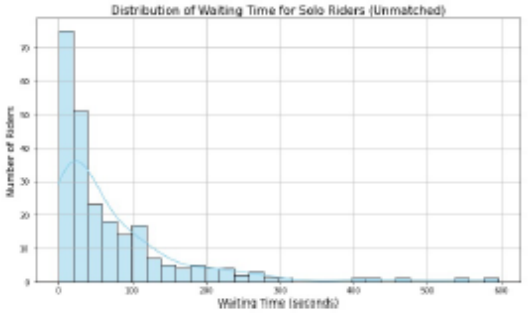
$$R \mid x, k \sim \text{Bernoulli}(\theta_{x,k})$$

- Expected profit:

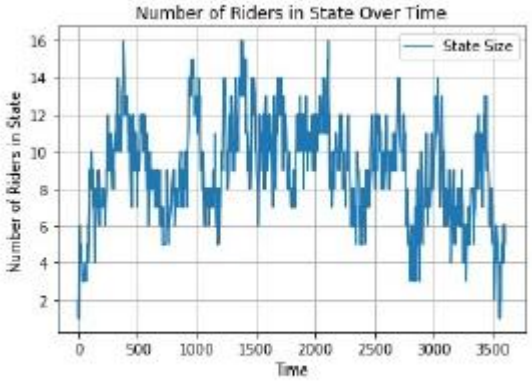
$$\Pi(x, k) = \mathbb{E}[R] \cdot (q_k - c) \cdot \ell = \theta_{x,k}(q_k - c)\ell$$



(a) Waiting time of converted riders.



(b) Waiting time of non-converted riders.



(c) Number of riders in the state.

Implementation – Pricing policy

Bayesian Priors and Posterior Structure

- Each context–arm pair (x, k) is assigned a Beta prior:

$$\theta_{x,k} \sim \text{Beta}(\alpha_{x,k}, \beta_{x,k})$$

- Maintain **3 levels of priors**:

1. Full-context: $(\alpha_{x,k}, \beta_{x,k})$
2. Pickup-only: $(\alpha^p_{p,k}, \beta^p_{p,k})$
3. Dropoff-only: $(\alpha^d_{d,k}, \beta^d_{d,k})$
4. Global: (α^g_k, β^g_k)

- All initialised to 1

Implementation – Pricing policy

Action Rule (Thompson Sampling)

- Sample acceptance rate for each arm k given context x:

$$\tilde{\theta}_k \sim \begin{cases} \text{Beta}(\alpha_{x,k}, \beta_{x,k}) & \text{if } (x,k) \text{ seen} \\ \text{Beta}\left(\frac{\alpha^p_{p,k}+\alpha^d_{d,k}}{2}, \frac{\beta^p_{p,k}+\beta^d_{d,k}}{2}\right) & \text{otherwise} \end{cases}$$

- Select arm:

$$k^\star = \arg\max_k \tilde{\theta}_k \quad \text{(or average of top 2 arms)}$$

- Deploy quote $q_{k^\star}$ = midpoint of selected bin

Offline Posterior Update

- After observing Bernoulli outcome $r \in \{0, 1\}$:

$$\alpha_{x,k^\star} \leftarrow \alpha_{x,k^\star} + r, \quad \beta_{x,k^\star} \leftarrow \beta_{x,k^\star} + 1 - r$$

- Apply same update to pickup-only, dropoff-only, and global priors

Implementation – Pricing policy

Benefits

- Analytic updates in $O(1)$ per decision
- Balances:
  - **Exploitation** of known arms
  - **Exploration** in sparse contexts via fallback priors
- Preserves accuracy while maintaining **real-time performance**

Implementation – Pricing policy

Online Decision (No Test-Time Learning)

- Posterior counts $\{\alpha_{x,k}, \beta_{x,k}\}$ are **frozen during evaluation**
- For each new rider with context $x = (p, d, s_b, w_b)$:

Case 1: Exact Cell Available

- Draw Thompson samples:

$$\tilde{\theta}_k \sim \mathrm{Beta}(\alpha_{x,k}, \beta_{x,k})$$

- Quote:

$$q = \frac{1}{2}\left(q_{(k_1)} + q_{(k_2)}\right)$$

where k_1, k_2 are top sampled arms

Case 2: Exact Cell Empty

- Average pickup & dropoff priors:

$$\bar{\alpha}_k = \tfrac{1}{2}(\alpha^p_{p,k} + \alpha^d_{d,k}), \quad \bar{\beta}_k = \tfrac{1}{2}(\beta^p_{p,k} + \beta^d_{d,k})$$

- Sample $\tilde{\theta}_k \sim \mathrm{Beta}(\bar{\alpha}_k, \bar{\beta}_k)$

Implementation – Pricing policy

Theoretical Guarantee (Contextual Thompson Sampling)

- Fix time horizon T, number of arms K
- For each (x, k), assume acceptance rate $\theta_{x,k} \sim \mathrm{Beta}(1, 1)$

Then:

$$\mathbb{E}[\mathrm{Regret}(T)] = \widetilde{O}(\sqrt{KT})$$

- Guarantees **sublinear regret** and stable performance even without test-time learning
- Enables exploration during training, while staying **Bayes-predictive** during deployment

Implementation – Pricing policy

Performance Summary

- Achieved **43% conversion rate** on held-out test weeks
- Price strategy finds effective trade-off between:
  - Acceptance
  - Margin
  - Queue control
- Hierarchical smoothing (pickup, dropoff, global) ensures price **stability** in sparse contexts
- Ready for live deployment with **no additional infrastructure**

Implementation – Matching policy

System-Wide Objective

- Accept a match (r, w) **only if** joint profit exceeds any solo or existing pairing alternative

Incremental Profit of Pairing

For rider r arriving at time t, and pool $\mathcal{S}(t)$, compute:

$$\Pi_{r,w} = q_r \ell_r + q_w \ell_w - c\ell_{rw} \quad \text{with } c = 0.70\$/\text{mi}$$

- ℓ_{rw}: shared route distance
- q_r, q_w: offline-estimated prices
- ℓ_r, ℓ_w: solo distances

Implementation – Matching policy

Best Alternative Profit (per waiting rider)

$$\mathrm{BestAlt}(w) = \max_{u \in \mathcal{S}(t) \setminus \{w\}} [q_w \ell_w + q_u \ell_u - c\ell_{wu}]$$

- Represents best known profit from matching w with anyone else
- Updated on each arrival:

$$\mathrm{BestAlt}(w) \leftarrow \max\{\mathrm{BestAlt}(w), \Pi_{r,w}\}$$

Matching Rule: Marginal Gain

$$\mathrm{Gain}_{r,w} = \Pi_{r,w} - \mathrm{BestAlt}(w)$$

- Match only if $\mathrm{Gain}_{r,w} > 0$
- Ensures every match **strictly improves** expected profit
- Prevents greedy matching from stealing from future opportunities
- Update cost remains $O(1)$

Implementation – Matching policy

Base Price Integration

- q_r, q_w drawn from base price lookup table
- Ensures consistency with pricing policy
- Prevents overestimation of pairing profit

Performance and Outlook

- Rooted in **supply chain principles**:
  - Shared cost efficiency
  - Queue-aware matching
- Guarantees:
  - Real-time feasibility
  - Stable match decisions
  - Compatibility with broader pricing framework

Implementation – Matching policy

Algorithm 2 Marginal-Gain Greedy Matching

Require: incoming rider r, current pool $\mathcal{S}(t)$
Ensure: partner $w^\star$ or `None`

if $\mathcal{S}(t) = \varnothing$ **then return** `None`
end if
$\mathcal{P} \leftarrow$ riders in $\mathcal{S}(t)$ whose *waiting time* $< 180\,\mathrm{s}$
$bestGain \leftarrow 0, \quad w^\star \leftarrow$ `None`
for $w \in \mathcal{P}$ **do**
    $\Pi_{r,w} \leftarrow q_r\, l_r + q_w\, l_w - c\, l_{rw}$ ▷ shared-trip profit
    $G_{r,w} \leftarrow \Pi_{r,w} - \mathrm{BestAlt}(w)$ ▷ marginal gain
    $\mathrm{BestAlt}(w) \leftarrow \max\{\mathrm{BestAlt}(w), \Pi_{r,w}\}$
    if $G_{r,w} > bestGain$ **then**
        $bestGain \leftarrow G_{r,w}, \quad w^\star \leftarrow w$
    end if
end for
return $w^\star$ **if** $bestGain > 0$ **else** `None`

Implementation – Matching policy

Marginal-Gain Matching Reformulated

- At decision time, define graph $G = (\mathcal{P}, E, g)$
  - $\mathcal{P}$: waiting rider set
  - $g_{ij} = (q_i \ell_i + q_j \ell_j - c\ell_{ij}) - \mathrm{BestAlt}(j)$
- Each g_{ij} measures **true marginal gain** from pairing i and j
- Edge added **only if both riders are unmatched**

Implementation – Matching policy

Greedy Matching Algorithm

- Scan all eligible edges in **descending order** of g_{ij}
- Add edge if both endpoints are free
- Classical result (Avis 1983, Pitts & Sung 2020):

$$\sum_{(i,j)\in\mathcal{M}_{\text{Greedy}}} g_{ij} \geq \frac{1}{2} \sum_{(i,j)\in\mathcal{M}_{\text{OPT}}} g_{ij} \qquad \text{(M1')}$$

- Guarantees **at least 50%** of optimal attainable profit
  - Within 180-second sliding window
  - Runtime: $O(|\mathcal{P}|) \leq O(180)$
  - Latency: **< 8 ms**

Implementation – Matching policy

Combined Guarantee

- Combines with:
  - Sublinear regret of pricing bandit
  - Matching approximation ratio
- Together, yield:
  - **Monotonic cumulative revenue growth**
  - Robust empirical performance
  - Near-optimality within operational constraints

Computational Results

Feature	Value	Insight
Profitability	$0.65/mi	Positive net margin, 68% revenue gain over baseline
Throughput	12.5 req/min	Handles high arrival rate under real-time latency constraints
Conversion Rate	43%	Context-aware pricing balances acceptance vs. margin
Matching Efficiency	54%	Marginal-gain matcher yields net-positive pairings
Cost Management	+15%	Distance caps & detour < 8% of solo miles
Price Structure	$0.62/mi	Adaptive bandits reduce low-value quotes while preserving acceptance
Responsiveness	56 sec avg	Median runtime 3 ms, 95th percentile under 5 ms (< 50 ms budget)

Computational Results

- **Trade-off navigated**:
  - Low prices ↑ conversion ↓ profit
  - High prices ↑ margin ↓ acceptance
- Solution: Thompson Sampling + context-based bandits balance both
- **Matching Policy Strength**:
  - Avoids overmatching
  - Guarantees only profit-improving pairings via marginal-gain test
- **System Integration**:
  - Shared pricing + dispatch logic
  - Fully interpretable, scalable, and operationally feasible

Calyber combines Bayesian pricing with greedy, state-aware matching to deliver a **fast**, **profitable**, and **data-driven** ride-sharing engine.