A Hybrid Approach to Rideshare Fleet Sizing: Integrating M/M/s Queueing Models with Agent-Based Simulation

Jiahe Ling[1] Thomas Flamand[1] Sarah Sheng[1] Alina Huang[1] Michael Zhang[1]
jl7264@columbia.edu tf2636@columbia.edu ys3002@columbia.edu yh3833@columbia.edu mz3160@columbia.edu

[1]*Department of Industrial Engineering and Operations Research, Columbia University*

1 Introduction

The Columbia Evening Shuttle service provides essential nighttime transportation for students, faculty, and staff, supporting safe and reliable mobility across campus. As demand for the service continues to grow, increasing pressure on driver availability during peak periods has led to longer passenger wait times and reduced service quality. Addressing these challenges requires a rigorous, data-driven assessment of **how adjustments to driver staffing levels and shift schedules influence overall system performance**. Using historical request data provided by Via, this study develops an $M/M/s$ queuing model alongside an agent-based simulation to represent the dynamic behavior of the shuttle network under alternative driver configurations. Average passenger wait time, total time in system, and driver utilization are examined as key operational metrics. By quantifying the performance effects of increasing the number of drivers during peak hours, the analysis aims to provide actionable recommendations for enhancing the efficiency and reliability of the Columbia Evening Shuttle service, ultimately improving outcomes for both riders and drivers.

2 Methodology

2.1 $M/M/m$ Model

2.1.1 Assumptions

This study applies the $M/M/s$ queueing framework under several standard operational assumptions. First, passenger arrivals within each weekday–hour (d, h) are assumed to follow a stationary Poisson process, reflecting the memoryless and independent nature of ride requests at this temporal resolution. Second, service times are modeled as independent exponential random variables with rate $\mu_{d,h}$, which captures the variability in trip durations while preserving analytical tractability. Third, each driver functions as an identical parallel server, and the system is spatially aggregated so that any available driver can serve any arriving request without repositioning delay. Finally, the number of active drivers $m_{d,h}$ is treated as fixed within each hour, producing a steady-state $M/M/s$ system for which classical performance measures can be computed.

2.1.2 Models

Given these assumptions, each weekday–hour (d, h) is modeled as an $M/M/s$ queue with arrival rate $\lambda_{d,h}$, computed by normalizing observed request counts by the number of occurrences of each weekday in October, and service rate $\mu_{d,h} = 1/(\text{mean trip duration})$ based on completed rides in the same period. The offered load is $a_{d,h} = \lambda_{d,h}/\mu_{d,h}$ and the utilization per server is $\rho_{d,h} = a_{d,h}/m_{d,h}$. The steady-

state probability that the system is empty is $P_{0,d,h} = \left[\sum_{k=0}^{m_{d,h}-1} a_{d,h}^{k}/k! + a_{d,h}^{m_{d,h}}/\{m_{d,h}!\,(1-\rho_{d,h})\}\right]^{-1}$. The probability that an arriving passenger must wait is $P_{\text{wait},d,h} = P_{0,d,h}\, a_{d,h}^{m_{d,h}}/\{m_{d,h}!\,(1-\rho_{d,h})\}$. The expected queue length is $L_{q,d,h} = P_{0,d,h}\, a_{d,h}^{m_{d,h}} \rho_{d,h}/\{m_{d,h}!\,(1-\rho_{d,h})^2\}$, which gives an expected waiting time in queue of $W_{q,d,h} = L_{q,d,h}/\lambda_{d,h}$ and a total expected time in system of $W_{d,h} = W_{q,d,h} + 1/\mu_{d,h}$. Appendix B provides concise derivations of these $M/M/s$ expressions.

2.2 The Simulation Framework (Agent-Based Modeling)

Although the $M/M/m$ model offers analytical convenience, it relies on strong simplifying assumptions. As a spatially homogeneous system, it accounts only for queueing and service-time delays and assumes that all drivers are equally close to all requests, that pickup travel time is negligible, that any driver can serve any request instantly, and that congestion or routing constraints do not exist. To better capture real rider request patterns and service times, an agent-based simulation is developed to relax these assumptions.

2.2.1 Modeling Assumptions

The Spatio-Temporal Agent-Based Rideshare Simulator (STARS) is built on the following assumptions:

- **Network:** The physical environment is the directed road network of Manhattan extracted from OpenStreetMap, and pickup/drop-off points are snapped to the nearest nodes.
- **Travel times:** Travel times are deterministic and derived from an hourly, piecewise-constant speed schedule based on New York City mobility statistics. In each hour, all vehicles share a uniform speed.
- **Demand:** Passenger requests are replayed from historical data. Requests are never canceled and remain in the system until served.
- **Service.** Each request is served by a single vehicle without transfers; boarding and alighting are assumed instantaneous, with no explicit stop service time.
- **Fleet:** Drivers are homogeneous, each operating a 4-seat vehicle and starting their shift from a fixed location at $(40.7505, -73.9934)$. Information and control. The dispatcher has complete information on all vehicles (location, load, routes) and all pending requests.
- **Service quality:** Vehicle capacity limits are enforced at all times, and any assignment that increases an existing passenger's promised in-system time beyond a preset percentage threshold is rejected.

2.2.2 Agent-Based Simulation Framework

STARS models each driver as an autonomous agent on the network and each request as an object with four states (pending, accepted, on-board, completed). The simulation proceeds in one-minute increments. At initialization, the road network is loaded, logging begins, and the fleet is instantiated according to the staffing profile. Drivers maintain their location, load, active route, and assigned requests; each request stores expected and actual pickup/drop-off times.

Routing. For each candidate assignment, the simulator solves an open pickup-and-delivery Traveling Salesman Problem (TSP) with precedence and capacity constraints. Let $V = \{0, \dots, n-1\}$ be the driver location and stops, d_{ij} the projected distances, $x_{ij} \in \{0, 1\}$ routing variables, u_i visit order, l_i vehicle load, $p(k), d(k)$ pickup/drop-off nodes for request k, q_i the change in the number of passengers at node i, initial load L_0, and capacity C.
A feasible solution is converted into a drivable route by connecting consecutive stops via shortest paths and converting lengths to times using the hour-specific speed. The route is accepted only if it respects capacity and does not exceed the allowed in-system delay for existing passengers.

Dispatching. The simulation advances in one-minute increments. At each step, the system (1) updates all drivers' positions along their assigned routes and executes any pickups or drop-offs whose scheduled arrival times have passed, marking drivers idle once their routes complete; (2) performs hourly fleet regulation by comparing the number of drivers currently accepting orders to the staffing profile, adding new drivers when under-supplied and retiring the longest-worked drivers when over-supplied; (3) ingests all new passenger requests whose timestamps fall within the current minute; and (4) evaluates each pending request across all active drivers, assigning it to the feasible driver offering the earliest pickup time and updating expected times accordingly. The clock is then advanced, and the process repeats until the end of the horizon. Throughout the simulation, driver and request events are logged, enabling subsequent computation of wait times, in-system times, and driver utilization.

$$
\begin{aligned}
\min_{x} \quad & \sum_{i,j} d_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{j \neq 0} x_{0j} = 1, \quad \sum_{i} x_{i0} = 0, \\
& \sum_{i \neq j} x_{ij} = 1 \ \forall j \neq 0, \quad \sum_{j \neq i} x_{ij} \leq 1 \ \forall i \neq 0, \\
& \sum_{i,j} x_{ij} = n - 1, \\
& u_0 = 0, \quad u_i - u_j + n x_{ij} \leq n - 1 \ \forall i \neq j, \\
& u_{p(k)} \leq u_{d(k)} - 1 \ \forall k, \\
& l_0 = L_0, \quad l_j = l_i + q_j \text{ if } x_{ij} = 1, \\
& 0 \leq l_j \leq C \ \forall j, \quad x_{ij} \in \{0, 1\}.
\end{aligned}
$$

Real Data Comparison. A comparison of 2025 completed trips from October 6 to 12 (18:00–22:59) shows that simulated performance substantially understates real-world delays. Real mean wait time is 5.39 times higher than simulated (20.39 vs. 3.79 minutes), in-system time is 3.39 times higher (28.96 vs. 8.53 minutes), and service time is 1.81 times higher (8.57 vs. 4.75 minutes). These scaling factors are not proportional, indicating that the real system is not uniformly slower: the largest gap occurs in waiting rather than travel time. This pattern suggests that the simulation underestimates congestion, driver availability, and operational frictions more than it underestimates roadway speeds. Nevertheless, the simulation remains appropriate for evaluating changes in staffing because it preserves the structural relationships between demand, fleet size, and system dynamics, even if absolute performance levels are optimistic.

3 Results

3.1 $M/M/m$ Model

This study models the relationship between average waiting time and the number of active drivers using the exact $M/M/s$ expression for the expected waiting time in queue, given by $W_q(s) = P_{\text{wait}}(s)/(s\mu - \lambda)$, where $P_{\text{wait}}(s)$ is the Erlang–C probability that all servers are busy. Although the function $W_q(s)$ decreases

rapidly with s, its dependence on s involves factorial and summation terms that prevent a closed-form solution for the staffing threshold. Consequently, the theoretical threshold must be defined as $x_\beta^{(\text{MMs})} = \min\{s : (W_q(s-1) - W_q(s))/W_q(s-1) < \beta\}$, where $\beta = 0.05$ specifies the desired marginal improvement criterion. Because $x_\beta^{(\text{MMs})}$ cannot be solved analytically, this study evaluates $W_q(s)$ numerically for successive integer values of s and then approximates the resulting discrete decay curve by fitting, for each date and hour, an exponential function of the form $W(x) = ae^{-bx} + c$. This smooth approximation captures both the rapid reduction in waiting time at low staffing levels and the diminishing returns that occur as s increases. Under this fitted model, the threshold condition $(W(x-1) - W(x))/W(x-1) < \beta$ yields the closed-form expression $x_\beta = \lceil (1/b)\ \ln(a(e^b(1-\beta) - 1)/(\beta c)) \rceil$, which provides a computationally efficient estimate of the staffing level at which further driver additions reduce expected waiting time by less than five percent.

Table 1 reports the resulting arrival rates λ and staffing thresholds $x_{\beta=0.05}$ for October 06 through October 12, 2025, covering the evening window from 18:00 to 22:59. Arrival rates at 18:00 range from approximately 57 to 76 across the week, then rise sharply on weekdays at 19:00, reaching values between 93 and 112 on October 06, October 07, October 09, and October 10. These high levels persist into 20:00 on several days, for example $\lambda = 108$ on October 06 and $\lambda = 107$ on October 10. Demand at 21:00 and 22:00 remains elevated on weekdays but falls noticeably on October 11 and October 12, where λ frequently lies between 67 and 82. The lowest hourly rate appears on October 11 at 23:00 with $\lambda = 48$, indicating substantially lighter weekend demand. Figure 3 summarizes how waiting times vary with staffing levels across the week.

Table 1: Hourly arrival rates λ and corresponding staffing thresholds for each day during October 6–12, 2025.

Hour	2025-10-06		2025-10-07		2025-10-08		2025-10-09		2025-10-10		2025-10-11		2025-10-12	
	λ	$x_{\beta=0.05}$	λ	$x_{\beta=0.05}$	λ	$x_{\beta=0.05}$	λ	$x_{\beta=0.05}$	λ	$x_{\beta=0.05}$	λ	$x_{\beta=0.05}$	λ	$x_{\beta=0.05}$
18	70	25	76	28	74	29	64	28	68	29	57	23	84	22
19	97	29	112	29	93	30	107	31	74	24	70	22	92	26
20	108	32	–	–	–	–	–	–	107	31	82	28	82	31
21	–	–	94	33	–	–	–	–	92	31	81	29	80	26
22	64	27	89	30	78	31	91	32	84	30	73	28	67	22
23	72	27	77	24	68	25	71	27	72	30	48	24	–	–

The staffing thresholds exhibit a strong positive association with the arrival rates. High-demand hours such as 19:00 to 21:00 on October 07, October 09, and October 10 yield x_β values between 29 and 33, implying that roughly thirty drivers are needed before additional staffing provides negligible improvement. Moderate-demand hours such as 18:00 or 22:00 on most days yield thresholds between 26 and 30. In contrast, the light-demand periods on October 11 require much smaller staffing levels, with x_β equal to 22 or 23 at 18:00 and 19:00, and equal to 24 at 23:00. These results show that the system reaches the diminishing-returns region quickly on low-demand evenings but only after substantial staffing on high-demand weekdays.

Some variability in x_β for similar values of λ reflects differences in the fitted parameters a, b, and c, which determine how sharply $W(x)$ declines with additional drivers. For example, October 10 at 19:00 exhibits a moderate arrival rate of $\lambda = 74$ but yields a relatively low threshold of $x_\beta = 24$, indicating a steeper decay in the fitted waiting-time curve compared with other hours. Such differences emphasize that x_β reflects both demand intensity and the empirical responsiveness of waiting times to additional staffing. Overall, the table reveals substantial weekday evening peaks, weaker weekend demand, and proportional variation in the staffing needed to reach diminishing returns.

3.2 The Simulation Framework (Agent-Based Modeling)

3.2.1 Number of Drivers vs. Wait Time

Figure 1 summarizes how system performance varies with the number of drivers. The average wait time exhibits a clear exponential decay, decreasing rapidly between 15 and 25 drivers before flattening beyond 30, consistent with strong diminishing returns in additional fleet capacity. Service time and in-system time also decline with more drivers, though with smaller magnitudes, indicating that increased fleet size primarily alleviates pickup delays rather than in-vehicle travel time. Pickup and drop-off estimation errors decrease steadily as well, reflecting improved route stability and reduced scheduling interference under higher driver availability. These results are based on simulations calibrated to the busiest hour of the week, October 9th from 20:00 to 20:59, which saw 218 requests and thus represent system behavior under the most demand-intensive conditions.

The fitted exponential model for average wait time, $W(x) = ae^{-bx} + c$ with $a = 3.7920$, $b = 0.09486$, and $c = 0.83810$, which has $R^2 = 0.9767$, provides a tight representation of the decay pattern in Figure 1. To identify when additional drivers yield negligible benefit, the smallest x is sought such that the relative improvement $(W(x-1) - W(x))/W(x-1) < \beta$. As derived in Appendix C, the closed-form solution is $x_\beta = \left\lceil \frac{1}{b} \ln\left(\frac{a(e^b(1-\beta)-1)}{\beta c}\right) \right\rceil$. Substituting the estimated parameters yields $x_{0.05} = 15$ for $\beta = 5\%$ and $x_{0.01} = 39$ for $\beta = 1\%$, where further increases in fleet size deliver strongly diminishing returns.

3.2.2 Effect of Driver Shift Duration

Figure 2 shows how performance varies with driver shift duration using data from October 6 between 18:00 and 22:59. Average wait time rises sharply from 1 to 2 hours and then plateaus, closely matching the fitted power-law model ($y = -0.499\,x^{-2.721} + 3.831$, $R^2 = 0.995$). Service time and in-system time increase slightly with longer shifts, indicating that extended driver availability primarily adds route carryover rather than improving throughput. Pickup and drop-off estimation errors also rise modestly, reflecting reduced schedule freshness over longer horizons.

3.3 Comparing M/M/s and STARS Recommendations

Comparing the $M/M/s$ predictions with the simulation results reveals several systematic differences in both the shape and scale of the waiting-time curves. The $M/M/s$ model, which assumes instantaneous pickup, spatial homogeneity, and no routing or congestion effects, predicts a much steeper decline in $W_q(s)$ and therefore smaller staffing thresholds: for example, many weekday evening hours in Table 1 reach $x_\beta = 26$–33 even when λ exceeds 100. In contrast, the simulation indicates that substantially more drivers are required before reaching diminishing returns. For the busiest hour of the week (October 9, 20:00–20:59), the simulated waiting-time curve falls rapidly only between 15 and 25 drivers and then flattens near 30, and the fitted exponential model yields $x_{0.05} = 15$ and $x_{0.01} = 39$, both markedly different from the corresponding $M/M/s$ thresholds. This divergence reflects phenomena absent from the queueing model but present in the simulation: pickup travel time, geographic imbalance, batching and repositioning delays, and residual congestion even when many drivers are available. As a result, the $M/M/s$ model consistently provides an optimistic lower bound on required staffing, while the simulation captures the higher real-world fleet levels needed to achieve comparable waiting-time reductions.

4 Conclusion

This study combines an analytical queueing model and an agent-based simulation to assess how driver staffing levels affect passenger waiting time. The $M/M/s$ model provides a simple theoretical benchmark, showing how hourly demand patterns translate into congestion and the staffing levels at which marginal improvements become small. During October 6 to 12, weekday evenings exhibit pronounced demand peaks, producing staffing thresholds in the mid-twenties to low-thirties, while weekend evenings require fewer drivers. These analytical thresholds, however, rely on idealized assumptions and therefore represent optimistic lower bounds.

The agent-based simulation offers a more realistic assessment by incorporating the Manhattan street network, spatial variation in driver locations, travel speeds, routing constraints, and dynamic assignment. Under these operational conditions, waiting times decline more slowly as the fleet grows, and substantial improvements occur only up to roughly 15 to 25 drivers before the system stabilizes near 30. The simulation shows that real delays are shaped by pickup travel time, geographic imbalance, and interactions among overlapping trips, which are not reflected in the analytical benchmark. As a result, higher staffing levels are required in practice than suggested by the $M/M/s$ model.

This study has several limitations. The analytical model abstracts from spatial structure and treats all drivers as equally available to all passengers. The simulation relies on historical demand from a single week, fixed driver behavior, and assumed routing and speed profiles. Future work could incorporate dynamic repositioning strategies, adaptive driver scheduling, and a broader set of demand scenarios to strengthen the generality of the findings.

GitHub Repository

The simulation code and analysis scripts used in this study are available at:
github.com/JiaheLing/Columbia_Shuttle.

Appendix

A Visualization

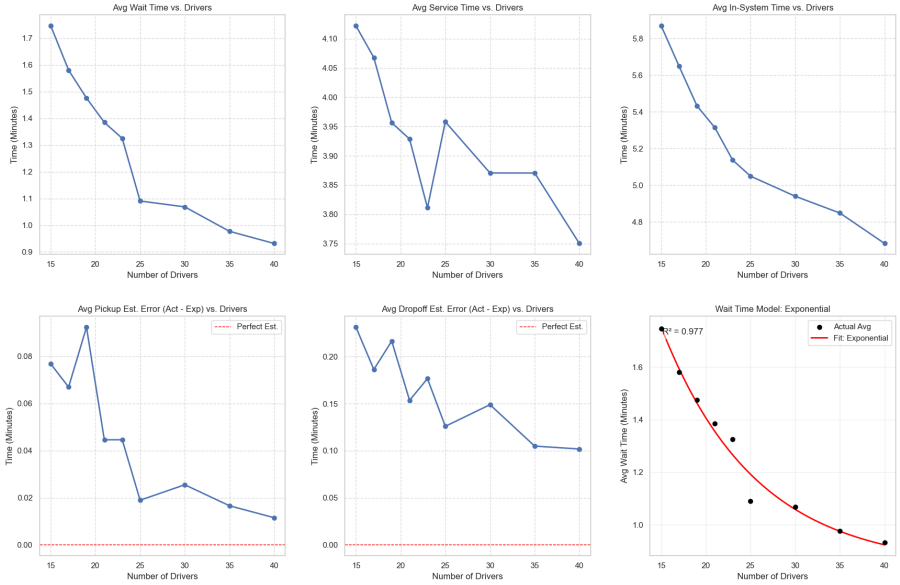


Figure 1: System performance metrics as a function of the number of drivers.

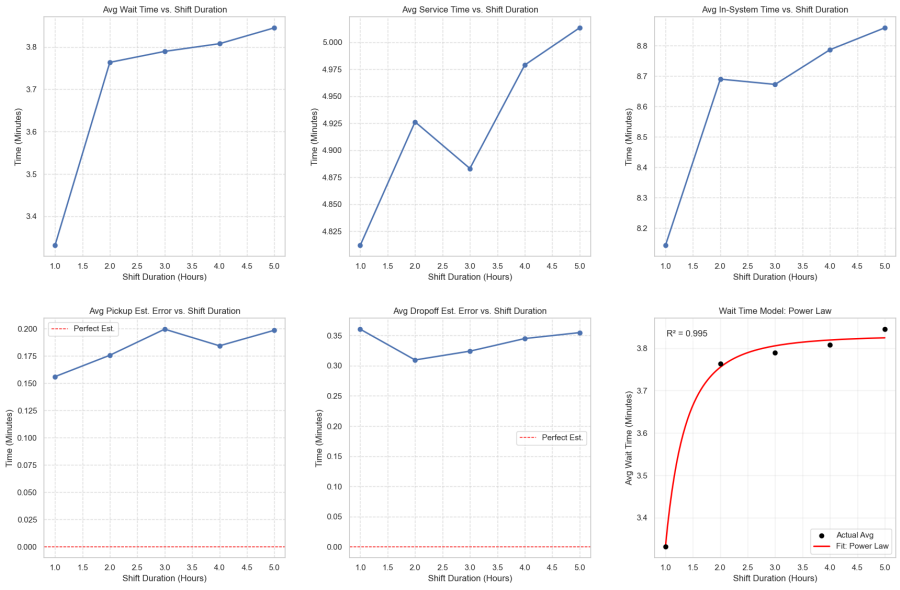
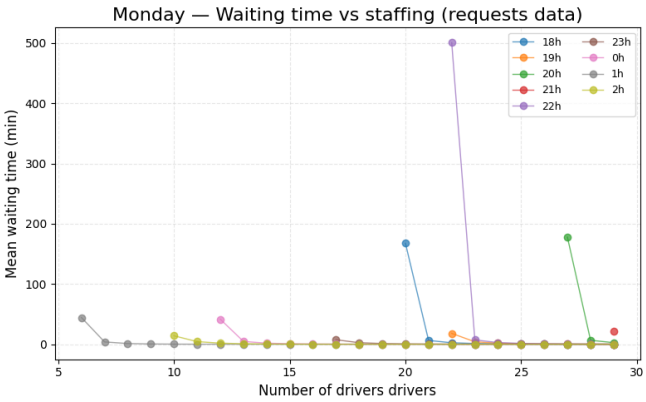
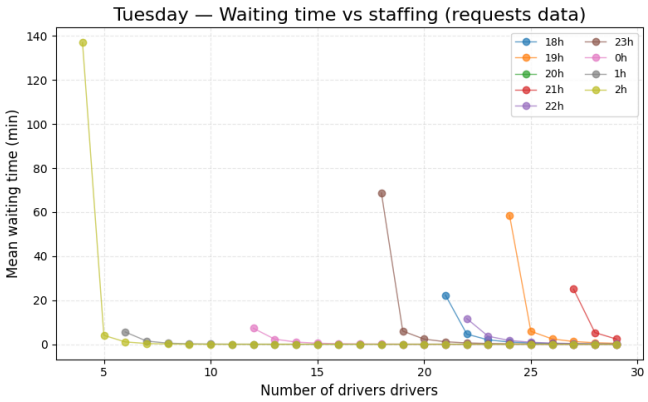


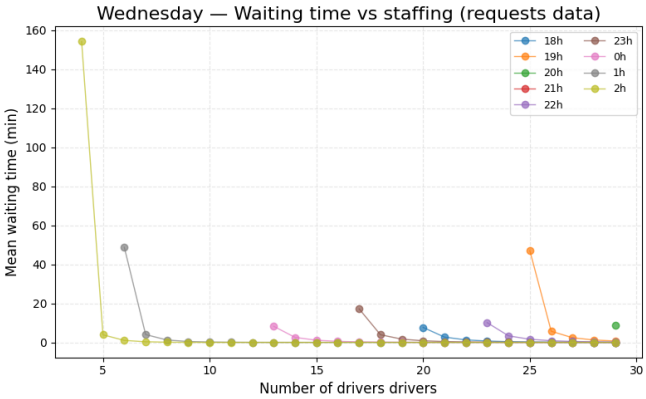
Figure 2: System performance metrics as a function of driver shift duration.



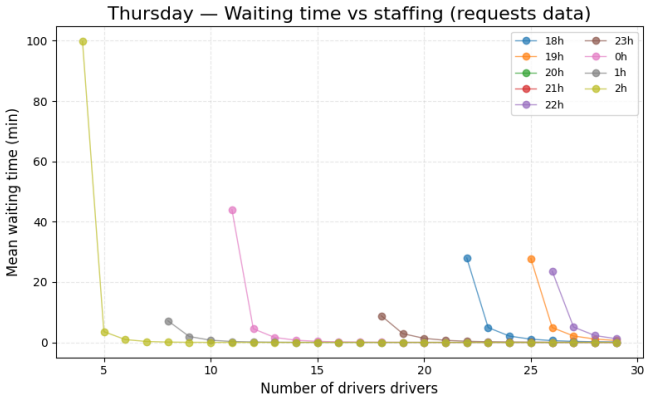
(a) Monday



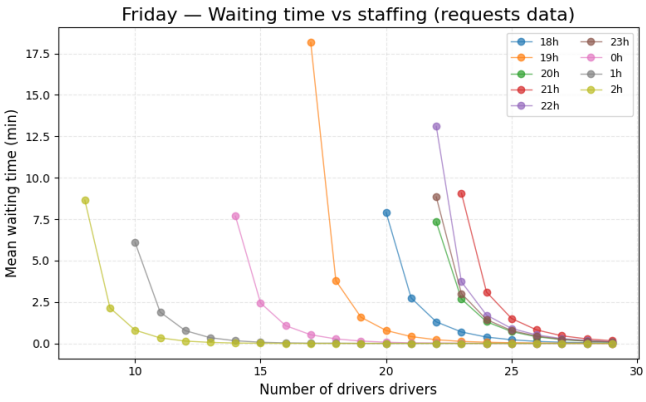
(b) Tuesday



(c) Wednesday



(d) Thursday



(e) Friday

(f) Saturday

(g) Sunday

Figure 3: Waiting time vs. staffing by weekday.

B M/M/s Derivations

Consider an $M/M/s$ system with arrival rate λ, service rate μ per server, and m servers. Define $a = \lambda/\mu$ and $\rho = \lambda/(m\mu) = a/m < 1$.

Birth–death balance gives, for $n \geq 1$,

$$\lambda P_{n-1} = \mu_n P_n, \quad \mu_n = \begin{cases} n\mu, & n \leq m, \\ m\mu, & n \geq m. \end{cases}$$

$$P_n = \frac{a^n}{n!} P_0, \quad 0 \leq n \leq m, \qquad P_n = \frac{a^m}{m!} \rho^{n-m} P_0, \quad n \geq m.$$

Normalization $\sum_{n=0}^{\infty} P_n = 1$ gives

$$1 = P_0 \sum_{n=0}^{m-1} \frac{a^n}{n!} + \frac{a^m}{m!} P_0 \sum_{k=0}^{\infty} \rho^k = P_0 \left[\sum_{n=0}^{m-1} \frac{a^n}{n!} + \frac{a^m}{m!(1-\rho)} \right],$$

$$P_0 = \left[\sum_{n=0}^{m-1} \frac{a^n}{n!} + \frac{a^m}{m!(1-\rho)} \right]^{-1}.$$

An arrival waits iff all m servers are busy:

$$P_{\text{wait}} = \sum_{n=m}^{\infty} P_n = P_m \sum_{k=0}^{\infty} \rho^k = \frac{P_m}{1-\rho} = \frac{P_0 a^m}{m!(1-\rho)}.$$

The queue length is $(N-m)^+$, so

$$L_q = \sum_{n=m+1}^{\infty} (n-m) P_n = \sum_{k=1}^{\infty} k P_{m+k} = P_m \sum_{k=1}^{\infty} k \rho^k.$$

Using $\sum_{k=1}^{\infty} k\rho^k = \rho/(1-\rho)^2$,

$$L_q = P_m \frac{\rho}{(1-\rho)^2} = \frac{P_0 a^m \rho}{m!(1-\rho)^2}.$$

The expected number in service is a, so

$$L = L_q + a.$$

By Little's Law, $L_q = \lambda W_q$, hence

$$W_q = \frac{L_q}{\lambda}, \qquad W = W_q + \frac{1}{\mu}.$$

C Driver Threshold Formula Derivation

The exponential wait-time model:

$$W(x) = ae^{-bx} + c.$$

The improvement condition:

$$\frac{W(x-1) - W(x)}{W(x-1)} < \beta, \qquad \beta \in (0,1).$$

$$W(x-1) = ae^{-b(x-1)} + c = ae^{-bx}e^{b} + c,$$

$$W(x-1) - W(x) = ae^{-bx}(e^{b} - 1).$$

Relative improvement:

$$\frac{W(x-1) - W(x)}{W(x-1)} = \frac{a(e^{b} - 1)}{ae^{b} + ce^{bx}}.$$

Imposing the threshold:

$$\frac{a(e^{b} - 1)}{ae^{b} + ce^{bx}} < \beta, \tag{1}$$

$$a(e^{b} - 1) < \beta(ae^{b} + ce^{bx}), \tag{2}$$

$$a(e^{b} - 1 - \beta e^{b}) < \beta c e^{bx}, \tag{3}$$

$$a\Big(e^{b}(1-\beta) - 1\Big) < \beta c e^{bx}. \tag{4}$$

Solving for x:

$$e^{bx} > \frac{a(e^{b}(1-\beta) - 1)}{\beta c}, \tag{5}$$

$$x > \frac{1}{b}\ln\left(\frac{a(e^{b}(1-\beta) - 1)}{\beta c}\right). \tag{6}$$

Minimum integer:

$$x_{\beta} = \left\lceil \frac{1}{b}\ln\left(\frac{a(e^{b}(1-\beta) - 1)}{\beta c}\right) \right\rceil.$$

D Spatio-Temporal Agent-Based Rideshare Simulator (STARS)

D.1 Class: `SimulationLogger`

Purpose: Manages file Input/Output streams to write simulation data to CSV files efficiently. It keeps file handles open to avoid the performance overhead of opening/closing files for every single event.

D.1.1 Attributes

Variable Name	Data Type	Description
`self.log_dir`	`str`	Directory path for logs (default: `"./logs/"`).
`self.driver_file`	`str`	Path to the driver log file.
`self.request_file`	`str`	Path to the request log file.
`self.d_writer`	`csv.writer`	CSV writer object linked to the driver log stream.
`self.r_writer`	`csv.writer`	CSV writer object linked to the request log stream.
`self.d_handle`	`TextIOWrapper`	Open file handle for the driver log.
`self.r_handle`	`TextIOWrapper`	Open file handle for the request log.

D.1.2 Functions

- **`initialize(self)`**
  - Creates the `./logs/` directory if it does not exist.
  - Opens files in write mode (`'w'`) and writes header rows.
- **`log_driver_event(self, driver_id, category, time, route_seq, capacity="")`**
  - Writes a row to `driver_logs.csv`.
  - **Logic:** Flushes the buffer immediately (`self.d_handle.flush()`) to ensure data is saved even if the script stops unexpectedly.
- **`log_request_summary(self, req)`**
  - Writes a row to `request_logs.csv` when a request is completed.
  - **Logic:** Flushes the buffer immediately.
- **`close(self)`**
  - Safely closes active file handles at the end of the simulation.

D.2 Class: `Stop`

Purpose: A lightweight data structure representing a specific physical node in the routing graph associated with a request.

D.2.1 Attributes

Variable Name	Data Type	Description
`self.request_id`	`str | int`	The ID of the request this stop belongs to.
`self.lat`, `self.lon`	`float`	Physical coordinates of the stop.
`self.is_pickup`	`bool`	`True` if Pickup, `False` if Drop-off. Critical for precedence constraints.
`self.count`	`int`	The number of passengers boarding/alighting at this node. Used by the solver to calculate dynamic load.

D.3 Class: `Request`

Purpose: Represents a passenger order and acts as a state machine tracking the order's lifecycle.

D.3.1 Attributes

Variable Name	Data Type	Description
`self.id`	`str`	Unique identifier.
`self.request_time`	`datetime`	Time the request entered the system.
`self.count`	`int`	Number of passengers.
`self.pickup_loc`	`Stop`	Origin node (initialized with `self.count`).
`self.dropoff_loc`	`Stop`	Destination node (initialized with `self.count`).
`self.status`	`str`	State: `'PENDING'`, `'ACCEPTED'`, `'ON_BOARD'`, `'COMPLETED'`.
`self.expected_..._time`	`datetime`	Estimated times set when assigned.
`self.actual_..._time`	`datetime`	Actual timestamps recorded by the Driver.

D.3.2 Functions

- **`update_expected_times(self, p_time, d_time)`**
  - **Logic:** Updates `expected_pickup_time` and `expected_dropoff_time` **only if they are currently `None`**. This preserves the original promise made to the passenger, allowing valid delay analysis later.
- **`record_pickup(self, time)`**
  - Sets `actual_pickup_time` and updates status to `'ON_BOARD'`.
- **`record_dropoff(self, time)`**
  - Sets `actual_dropoff_time` and updates status to `'COMPLETED'`.

D.4 Class: `Route`

Purpose: The computational engine handling **Optimization** (Gurobi TSP) and **Geometry** (Pathfinding).

D.4.1 Attributes

Variable Name	Data Type	Description
`self.G_latlon`, `self.G_proj`	`MultiDiGraph`	References to the map graphs (unprojected and projected).
`self.all_stops`	`List[Stop]`	Pool of locations to visit.
`self.start_coords`	`Tuple`	Starting location for this calculation.
`self.speed_mps`	`float`	Vehicle speed (m/s), derived from the dynamic speed passed in `__init__`.
`self.vehicle_capacity`	`int`	**Dynamic:** Max capacity of the specific vehicle.
`self.initial_load`	`int`	**Dynamic:** Passengers already on board at start.
`self.optimal_sequence`	`List[Stop]`	Ordered list of stops from the solver.
`self.stop_arrival_times`	`List[datetime]`	Calculated arrival times for the sequence.
`self.route_line`	`LineString`	Continuous geometric path.

D.4.2 Functions

- **`_solve_tsp_gurobi(self)`**
  - **Logic:** Mixed Integer Linear Programming (MILP) solver.
  - **Constraints:**
    1. **Flow:** Open TSP (Start $\rightarrow$ End, no return).
    2. **Precedence:** Pickup Index < Dropoff Index.
    3. **Initial Load:** $Load[0] == self.initial\_load$.
    4. **Capacity:** $Load[node] \leq self.vehicle\_capacity$.
    5. **Time Limit:** Hard limit of **60.0 seconds** to prevent freezing.
- **`generate_route(self)`**
  - **Logic:** Calls solver. If feasible, calculates `nx.shortest_path` between stops, stitches geometry, and calculates times. Returns `False` if solver fails or path is empty.
- **`get_location_at_time(self, query_time)`**
  - **Logic:** Linear interpolation along `route_line` based on elapsed time.

D.5 Class: `Driver`

Purpose: An autonomous agent that moves, manages capacity, and evaluates requests based on dynamic conditions.

D.5.1 Attributes

Variable Name	Data Type	Description
`self.id`	`str`	Unique Driver ID.
`self.current_location`	`Tuple`	Current (lat, lon).
`self.start_shift_time`	`datetime`	Time driver started shift.
`self.accepting_orders`	`bool`	`True` if within shift limits; `False` if retired.
`self.capacity`	`int`	**Dynamic:** Capacity assigned to this driver (e.g., 4 or 6).
`self.speed_schedule`	`Dict`	Reference to the global speed schedule.
`self.passengers_on_board`	`int`	Current passenger count.
`self.active_requests`	`List`	Requests currently assigned.
`self.route`	`Route`	Active route object. `None` if idle.

D.5.2 Functions

- **`_get_unvisited_stops(self, current_time)`**
  - **Logic:** Returns stops where the request action (pickup/dropoff) **has not happened yet**. This prevents race conditions where a stop at the exact current second might be skipped.
- **`status_update(self, current_time, logger)`**
  - **Logic:**
    1. Updates location.
    2. Checks for passed stops (time $\leq$ current).
    3. Updates `passengers_on_board` and calls request recording methods.
    4. Sets `is_idle=True` if route finishes.
    5. **Returns:** List of requests that became `COMPLETED` (for logging).
- **`request_validity_check(self, new_req, current_time)`**
  - **Logic:**
    1. **Speed Lookup:** Gets `current_speed` from `speed_schedule` based on hour.
    2. **Pre-check:** If `new_req.count > self.capacity`, reject.
    3. **Optimization:** Creates `Route` with `initial_load=passengers_on_board`.
    4. **Delay Check:** Rejects if existing passengers are delayed > 20%.
- **`accept_request(self, req, potential_route, logger, current_time)`**
  - **Logic:** Commits to the new route and updates `expected` times for all passengers.

D.6 7. Class: `Simulation`

Purpose: Master controller for time, fleet lifecycle, and dispatching.

D.6.1 Attributes

Variable Name	Data Type	Description
`self.driver_schedule`	`Dict`	{Hour: Target_Driver_Count}.
`self.shift_hours`	`int`	Max shift duration.
`self.vehicle_capacity_types`	`List[int]`	Pool of capacities (e.g., `[4, 6]`) for new drivers.
`self.pending_requests`	`List`	Queue of unassigned requests.
`self.vehicles`	`List`	Active fleet.

D.6.2 Functions

- **`add_drivers(self, count)`**
  - **Logic:** Spawns `count` drivers. Randomly assigns capacity from `vehicle_capacity_types` and location from pool.
- **`manage_fleet(self)`**
  - **Logic (Run Hourly):**
    1. **Check Shifts:** If `time_worked >= shift_hours`, set `accepting_orders = False`.
    2. **Upscale:** If active count < target, add drivers.
    3. **Downscale (Remove):** If active count > target, identify excess.
       - **Sort:** Longest worked (Descending) → ID (Ascending).
       - **Action:** Force retire top N drivers.
    4. **Cleanup:** Remove driver from list ONLY if `not accepting_orders` AND `is_idle`.
- **`find_best_vehicle(self, req)`**
  - **Logic:** Greedy dispatch. Returns the driver who can offer the **earliest pickup time**.
- **`run_simulation(self)`**
  - **Logic (Main Loop):**
    1. **Update Fleet (`status_update`):** Runs FIRST to ensure accurate state at the top of the minute.
    2. **Manage Fleet:** Runs hourly.
    3. **Ingest Requests:** Adds new requests to pending.
    4. **Assign:** Processes pending queue.
    5. **Step Time:** Increments clock.