# Introduction:

This project develops a configurable, text-based microsimulation (in python) of a four-arm roundabout to quantify how design and behavioral parameters affect operational efficiency. The simulator is simple, no graphics and single file, so that geometry (e.g., diameter, number of circulating lanes) and behavior (e.g., arrival rates, driver reaction time, critical gaps) can be varied easily and their impacts measured reliably and reproducibly. The current focus is the roundabout and maximizing its efficiency, as well as identifying breaking points in the system (e.g., the critical volume at which another lane or a larger diameter becomes necessary). A signalized intersection simulation with similar optimization goals will follow. The goal being to have direct comparisons between the two systems, and determine which is best, depending on the given demand.

In parallel, we will replicate these scenarios in SUMO (Simulation of Urban Mobility) for both the roundabout and signalized intersections, using the same modeling logic, comparable queuing rules at entries, consistent car-following dynamics, and aligned demand patterns, so results are methodologically comparable across platforms. Coding our own Python simulation gives us greater control and transparency over assumptions, algorithms, and metrics used, allowing for a better understanding of how each modeling choice influences outcomes. SUMO, in turn, provides a visual environment, richer built-in diagnostics/metrics, and a great ecosystem for scenario management. We will use SUMO both to augment our analysis (e.g., additional KPIs and visual inspection of flows) and to validate the Python results, treating agreement across platforms as evidence for model correctness and using any discrepancies to refine assumptions (e.g., gap-acceptance thresholds, reaction delays, lane-discipline rules). Together, the Python and SUMO tracks create a robust workflow for optimizing designs and testing sensitivity to demand, geometry, and driver behavior.

# Problem Statement & Challenges:

Urban traffic intersections operate as critical bottlenecks in transportation networks. Poor design or control strategies lead to:

- Capacity saturation: Queue divergence when demand exceeds service capacity
- Excessive delays: Reduced level-of-service affecting user satisfaction
- Safety concerns: Increased conflict points and crash risk
- Environmental impact: Elevated emissions from idling and stop-and-go behavior

The fundamental question: Given demand characteristics (volume, turning movements) and site constraints, what intersection control strategy and geometric configuration maximize efficiency while ensuring stability?

Technical Challenges

Stochastic Variability
Traffic is inherently random:
- Arrival processes: Poisson or more complex (platooning, signal influence)
- Driver heterogeneity: Critical gap acceptance varies by driver
- Turning choices: Probabilistic movement selection
- Behavioral uncertainty: Reaction times, desired speeds, aggressiveness

Solution: Monte Carlo simulation with sufficient replication to capture statistical properties.

Multi-Objective Optimization
No single "best" design exists; trade-offs include:
- Maximize throughput ↔ minimize delay
- Reduce emissions ↔ increase flow rate
- Ensure safety ↔ optimize capacity

Solution: Multi-objective optimization identifying Pareto-optimal configurations; decision-makers select based on priorities.

Continuous Dynamics with Discrete Events
Vehicles follow continuous ODEs (car-following, lane-changing) while experiencing discrete events (arrivals, merges, exits).

Solution: Hybrid simulation architecture with time-stepping for continuous dynamics and event scheduling for discrete transitions.

Reaction Delay (Non-Markovian Effects)
Human reaction time $\tau \approx 1–2$s means acceleration at time *t* depends on states at *t−τ*, creating delay-differential equations (DDEs).

Solution (Python): History buffer storing vehicle states; retrieve delayed snapshots for IDM calculation.
Solution (SUMO): `actionStepLength` approximates delayed responses (vehicles update control every $\tau$ seconds).

Gap Acceptance at Yield Points
Roundabout entries require modeling:
- Critical gap: first vehicle needs larger gap to merge (right-skewed distribution → lognormal)
- Follow-up headway: subsequent vehicles follow tighter (normal distribution)
- Platoon dynamics: spoiled opportunity breaks platoon; next attempt reverts to T<sub>c</sub>

Solution: Per-vehicle stochastic gap draws (Python) or threshold with impatience growth (SUMO).

Cross-Platform Consistency
Ensuring comparable results between Python and SUMO requires:
- Parameter mapping (e.g., lateral acceleration → ring speed limit)
- Equivalent arrival processes
- Matched car-following models
- Aligned failure criteria

Solution: Comprehensive parameter mapping; validation protocol comparing distributions, not just means.

# Mathematical/Computational Background:

Mathematical/Computational Background completed

- Roundabout: DDES, Poisson process, Exponential Distrobution, Stochastice modeling for seed,

What follows is the various mathematical and computational background needed to realize this project. We'll also demonstrate how we've implemented some of it in our python code, thus far, and how SUMO handles it as well.

**Roundabout Mathematical/Computational Background:**

1. Poisson arrivals

The roundabout vehicle arrival process (upstream and downstream) can be modelled by a **homogeneous Poisson process.** The one variable (customizable knob) associated with this model is the rate lamda (vehicles/second).

There are two different views.

**Counts view.** If you watch an approach for TTT seconds, the number of arrivals N(T)N(T)N(T) is random with mean E[N(T)]=λT\mathbb{E}[N(T)] = \lambda TE[N(T)]=λT. The distribution is Poisson, so variability matches the mean: Var[N(T)]=λT\mathrm{Var}[N(T)] = \lambda TVar[N(T)]=λT. That gives us the "sometimes a few more, sometimes a few less" effect.

**Gaps view.** The **time between successive vehicles** at the yield line (the interarrival) is random with an **exponential** distribution. This distribution is **memoryless**: after waiting t seconds with no car, the chance a car appears in the next instant is *unchanged*. The average interarrival is 1/λ seconds. Short gaps are common; long gaps happen, but more rarely.

These two views are mathematically equivalent: exponential interarrivals ⇔ Poisson counts.

**Equations.**

$$\Delta t \sim \mathrm{Exp}(\lambda), \qquad P(\Delta t > t) = e^{-\lambda t}, \qquad \mathbb{E}[\Delta t] = \tfrac{1}{\lambda}.$$

$$N(T) \sim \mathrm{Poisson}(\lambda T), \qquad \mathbb{E}[N(T)] = \lambda T, \quad \mathrm{Var}[N(T)] = \lambda T.$$

**Implementation in our code:**

```
def _next_arrival_time(self, arm: int, now: float) -> float:
    lam = max(1e-9, self.cfg.demand.arrivals[arm])   # λ (veh/s), safeguarded > 0
    return now + random.expovariate(lam)            # draw Δt ~ Exp(λ); next car at now+Δt
```

2. Turning choice as a categorical random variable (L/T/R)

Each arriving driver chooses a movement—Right, Through, or Left—according to given **probabilities** (pR,pT,pL)(p_R, p_T, p_L)(pR,pT,pL) that sum to 1. Sampling from such a **categorical** distribution is commonly done with an **inverse-CDF** trick: draw a uniform u∈[0,1)u\in[0,1)u∈[0,1) and see in which cumulative bucket u falls.

**Equations**.

Let the cumulative thresholds be $c_1 = p_R, c_2 = p_R + p_T, c_3 = 1$.

$$u < c_1 \Rightarrow \mathrm{Right}, \quad c_1 \le u < c_2 \Rightarrow \mathrm{Through}, \quad c_2 \le u < 1 \Rightarrow \mathrm{Left}.$$

**Implementation in our code:**

```
def _draw_turn_steps(self) -> int:
    L, T, R = self.cfg.demand.turning_LTR
    u = random.random()                  # U[0,1)
    if u < R:       return 1             # Right  (next exit)
    elif u < R + T: return 2             # Through(opposite)
    else:           return 3             # Left   (third exit)
```

3. Gap acceptance: first car's "critical gap" and platoon "follow-up headway":

At a yield line, the **first queued vehicle** must find a sufficiently large **critical time gap** TcT_cTc in the circulating stream to merge. Once that leader squeezes in, **subsequent vehicles** can often follow with a tighter **follow-up headway** TfT_fTf, creating a short platoon.

Realistically, drivers differ in their behaviour:

Tc should be **positive** and often **right-skewed** (most drivers accept 2–4 s; a minority want much larger). A **lognormal** fits this shape well.

TfT_fTf is more **symmetric** around a typical value (~2 s). A **normal** (with a small lower bound) is a simple approximation.

A merge happens when two conditions hold **at the same time**:

1. **Time condition:** the **time until the next circulating vehicle** reaches the entry, tnextt_\text{next}tnext, is at least the required headway TneededT_\text{needed}Tneeded (which is TcT_cTc for the first car; TfT_fTf while platooning).

2. **Space condition:** immediate **front and rear buffers** at the merge point exceed a minimum distance (comfort/safety).

If an approaching ring vehicle will arrive **before** the needed headway materializes (i.e., tnext<Tneededt_\text{next} < T_\text{needed}tnext<Tneeded), the developing opportunity is **spoiled**, the platoon is considered **broken**, and the next attempt reverts to **first-car** behavior.

A merge happens when two conditions hold **at the same time**:

1. **Time condition:** the **time until the next circulating vehicle** reaches the entry, tnextt_\text{next}tnext, is at least the required headway TneededT_\text{needed}Tneeded (which is TcT_cTc for the first car; TfT_fTf while platooning).

2. **Space condition:** immediate **front and rear buffers** at the merge point exceed a minimum distance (comfort/safety).

If an approaching ring vehicle will arrive **before** the needed headway materializes (i.e., tnext<Tneededt_\text{next} < T_\text{needed}tnext<Tneeded), the developing opportunity is **spoiled**, the platoon is considered **broken**, and the next attempt reverts to **first-car** behavior.

Equations.

- Sampling:

$$T_c \sim \text{Lognormal}(\mu, \sigma), \quad T_f \sim \mathcal{N}(\mu_f, \sigma_f) \text{ (truncated below)}.$$

- Acceptance test at time $t$:

$$\text{merge if} \quad t_{\text{next}} \geq T_{\text{needed}} \ \text{ and } \ d_{\text{ahead}}, d_{\text{behind}} \geq d_{\min}.$$

**Implementation in our code:**

**Defining/Initializing.**

```
def _draw_crit_gap(self) -> float:  # lognormal for T_c (positive, skewed)
    m, s = self.cfg.gaps.crit_gap_mean, self.cfg.gaps.crit_gap_sd
    mu = math.log((m*m) / math.sqrt(s*s + m*m))
    sigma = math.sqrt(math.log(1 + (s*s)/(m*m)))
    return max(0.2, random.lognormvariate(mu, sigma))


def _draw_followup(self) -> float:  # normal for T_f (bounded below)
    m, s = self.cfg.gaps.followup_mean, self.cfg.gaps.followup_sd
    return max(0.2, random.gauss(m, s))
```

**Function calls.**

```
# Decision
needed = self.next_needed_headway[i] or q[0].crit_gap  # T_needed = T_f (platoon) or T_c (first car)
t_next = self._time_to_nearest_ring_vehicle(self.entry_pos[i], lane_idx)
if t_next >= needed and self._space_ok_at_merge(lane_idx, self.entry_pos[i], self.cfg.driver.s0 + 2.0):
    # merge; set follow-up for the next car
    self.next_needed_headway[i] = v.followup
```

else:

if t_next < needed:  # opportunity spoiled by approaching ring car

self.next_needed_headway[i] = 0.0  # break platoon → next first car must meet T_c

4. Reaction delay (DDE-style):

Human drivers don't react instantaneously. A simple way to incorporate this is a **delay-differential-equation (DDE) surrogate**: compute the ego's (specific vehicle currently being being simulated) acceleration using the **states from t−τt-\taut−τ** (ego's own speed and the leader's speed/gap as of τ\tauτ seconds ago). This can damp unrealistic "instant" reactions and produce smoother, more realistic dynamics.

**Informal equation:**

$$a(t) = f\big(\text{ego}(t-\tau), \text{leader}(t-\tau)\big),$$

**Implementation in our code:**

# keep ~tau/dt snapshots of (pos,speed) for all vehicles

self.max_hist_len = int(math.ceil(max(0.1, cfg.driver.tau) / cfg.dt)) + 2

# on each step

self._push_history()  # save current (pos,speed) by vehicle id

# when updating

steps_back = int(round(self.cfg.driver.tau / dt))

snap = self._snapshot(steps_back)  # states from t-τ

pos_d, v_d = snap.get(v.id, (v.pos, v.speed)) # ego at t-τ

_, gap_d, vL_d = self._leader_at_delayed_time(lane, pos_d, snap)  # leader at t-τ

5. Intelligent Driver Model (IDM): continuous car-following law

The IDM gives a smooth acceleration $a$ that blends two effects:

- **Free-road desire** to reach a target speed $v_0$: $1 - (v/v_0)^{\delta}$.
  As $v \to v_0$, this term shrinks to 0; $\delta \approx 4$ shapes how sharply it fades.
- **Interaction (safety) with a leader** at gap $s$: $-(s^*/s)^2$, where the desired gap

$$s^* = s_0 + vT + \frac{v\,(v - v_{\text{lead}})}{2\sqrt{a_{\max} b}}.$$

  This increases with speed ($vT$) and with approaching rate ($v - v_{\text{lead}}$).

The final acceleration is:

$$a = a_{\max}\left(1 - \left(\frac{v}{v_0}\right)^{\delta} - \left(\frac{s^*}{s}\right)^2\right).$$

**Implementation in our code:**

s0, T, a_max, b, delta = drv.s0, drv.T, drv.a_max, drv.b_comf, drv.delta

dv = v_d - vL_d

s_star = s0 + v_d*T + (v_d*dv) / max(1e-6, 2.0*math.sqrt(a_max*b))

acc = a_max * (1.0 - (v_d / max(0.1, v0))**delta - (s_star / max(1e-3, gap_d))**2)

5. Time stepping (numerical integration):

The continuous dynamics (v˙=a\dot v = av˙=a, x˙=v\dot x = vx˙=v) are advanced in **small steps** Δt\Delta tΔt using a simple and robust **Euler** scheme. We clamp speed to [0,v0][0, v_0][0,v0] for physicality.

**Equation**.

$$v_{t+\Delta t} \approx \text{clamp}(v_t + a_t \Delta t,\, 0,\, v_0), \qquad x_{t+\Delta t} \approx x_t + v_{t+\Delta t}\Delta t.$$

v_new = clamp(v.speed + acc * dt, 0.0, v0)

x_new = (v.pos + v_new * dt) % self.C

Mathematical/Computational Background TODO

- RL (PPO), SUMO Math (discrete modelling), Signal Math Modeling

# Methodology:

(current code for text based roundabout simulation)

(SUMO code methodology)

# Assessment & Evaluation:

There are four key metrics that we will be using to measure simulation performance:

Per 5-min window (already in code) and for the hour:

- **Throughput (veh/hr):** $\text{thr} = \dfrac{\text{exits} \times 3600}{\text{window\_sec}}$.
- **Delay:** mean and **P95** entry delay (s/veh).
- **Queues:** max queue length per arm (veh).
- **Stability indicators:** exit/arrival ratio, time-to-steady behavior.

We shall use SUMO to verify simulation accuracy. (Jaathavan, you can expand on this, what will this look like + what else can SUMO do for us from an assessment/evaluation perspective.)

# References:


-https://en.wikipedia.org/wiki/Poisson_point_process
-https://en.wikipedia.org/wiki/Log-normal_distribution
-https://en.wikipedia.org/wiki/Normal_distribution
-https://en.wikipedia.org/wiki/Inverse_transform_sampling
-. Zheng, D., Chitturi, M. V., Bill, A. R., & Noyce, D. A. (2011). *Critical gaps and follow-up headways at congested roundabouts.* Midwest Regional University Transportation Center, University of Wisconsin–Madison. Retrieved from

https://topslab.wisc.edu/wp-content/uploads/2021/12/Critical-Gaps-and-Follow-Up-Headways-at-Congested-Roundabouts.pdf
-Akçelik, R. (2008). *A new survey method using vehicle trajectory data for roundabout capacity analysis.* SIDRA Solutions Technical Paper TP-08-01. Melbourne, Australia. Retrieved from https://www.sidrasolutions.com/media/782/download
-Mathematical Modelling - A Case Studies Approach (Dr. Bohun's textbook)
-https://en.wikipedia.org/wiki/Delay_differential_equation (DDE wiki)

# Timeline:

Completed (Weeks 1–6)

Week 1–2: Problem Scoping & Literature Review
- Reviewed roundabout capacity analysis method
- Identified gap acceptance models
- Selected IDM for car-following; DDE for reaction delay

Week 3–4: Python Microsimulation Development
- Implemented core simulation loop (`Roundabout.py`)
- Integrated Poisson arrivals, lognormal critical gaps, IDM with DDE
- Validated against theoretical benchmarks (e.g., M/M/1 queue for simple cases)

Week 5–6: SUMO Pipeline Development
- Created network generator (`generate_network.py`) using `netconvert`
- Implemented demand generator (`generate_routes.py`) with Poisson flows
- Developed TraCI-based simulation runner (`run_simulation.py`) with 5-minute windowing

Current Status (Week 6)
Text-Based Simulation: Python microsimulation framework operational
SUMO Infrastructure: Network generation and TraCI integration complete
In Progress: SUMO roundabout simulation validation and analysis pipeline

Planned (Weeks 7–12)

Week 7–8 (Oct. 8–22): Text Simulation Completion + Midterm Report + SUMO Roundabout Progress
- Complete analysis infrastructure (`analyze_results.py`) with failure detection
- Implement parameter sweep orchestrator (`optimize.py`) for automated grid search
- Execute initial cross-platform validation between Python and SUMO

- Midterm Report: Document Phase 1 methodology, mathematical background, and preliminary results

Week 9 (Oct. 23–29): Roundabout Failure Analysis + SUMO Completion
- Study failure points: queue divergence, capacity saturation, excessive delays
- Complete SUMO roundabout simulation with comprehensive metrics collection
- Generate visualization suite (`visualize_results.py`) for performance analysis

Week 10 (Oct. 30–Nov. 5): Roundabout Optimization Strategies
- Execute full 30-scenario parameter sweep (geometry × demand combinations)
- Identify optimal configurations for maximum throughput, minimum delay, best balance
- Determine geometric constraints and breaking points for roundabout effectiveness

Week 11 (Nov. 6–12): Signalized Intersection Development
- Adapt network generator for 4-way signalized intersections
- Implement Webster's Method for optimal fixed-time signal control
- Develop both text-based and SUMO signalized intersection simulations

Week 12 (Nov. 13–19): Signal Optimization & Advanced Control
- Study failure modes in signalized intersections vs. roundabouts
- Implement Proximal Policy Optimization (PPO) for adaptive signal control
- Compare performance: fixed-time vs. adaptive vs. roundabout strategies

Week 13–14 (Nov. 20–Dec. 2): Final Analysis & Documentation
- Week 13: Compile comprehensive findings, prepare final presentation
- Week 14: Complete final report with cross-platform validation, optimization results, and strategic recommendations

# Contributions: