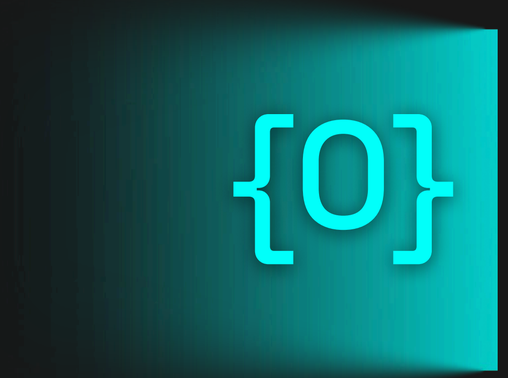


Information Asymmetry as Operational Failure

Designing and validating a Cognitive Decision support System for Multi-Node Service Networks

Table of Content

{#1} The Argument

The central question I designed this project around was precise: if you replaced proximity-based routing with cognitively realistic, information-guided routing, at what point does a structurally imbalanced service network stabilise - and what does that transition actually look like?

I constructed the entire experimental system to answer it. I configured two campus nodes - University Park and Jubilee - with asymmetric staffing distributions and spatial positions. I introduced a deliberate geographic demand bias, with 60% of agents spawning in the Jubilee-proximate region of the environment, reflecting the accommodation density reality of the real-world campus network. I set the arrival rate at 50 agents per hour as a deliberate stress condition. I designed three app adoption rate conditions at 0%, 50%, and 80% to isolate exactly how information availability changes system behaviour. None of these parameters were given. Each one was a design decision made to test a specific hypothesis: that the failure mode of a capacity-sufficient service network is informational, not operational, and that the correct intervention is architectural rather than organisational.

The architecture I built to test this was a hybrid of Discrete Event Simulation and Agent-Based Modelling. I chose this combination deliberately. A pure DES would model queue throughput accurately but could not explain why the queue forms in the first place. A pure ABM would model individual routing decisions but could not validate those decisions against credible SLA breach counts. The hybrid was the only architecture that could answer the actual question. The DES layer governed the operational environment - queue logic, staff resource pools, service time distributions, SLA auditing. The ABM layer governed individual student cognition - digital literacy, emotional state, social context, campus reputation memory. I connected the two layers through a Friction Transfer mechanism I designed specifically for this model: student frustration transfers to staff as emotional labour, staff fatigue degrades service throughput, slower throughput generates longer queues, longer queues produce more frustrated students. This feedback loop is absent from standard simulation models. I built it because validating a routing intervention without modelling the cascade it needs to break would have produced results that were technically correct and practically meaningless.

The experimental results confirmed the hypothesis across 1000 replications per phase. At 0% adoption - pure proximity routing with no information layer - the system produced a mean of 140.07 SLA breaches per shift with a standard deviation of 39.57. At 50% adoption the mean fell to 23.914 breaches with a standard deviation of 26.90 - an 82.9% reduction. At 80% adoption the mean reached 7.312 with a standard deviation of 12.68 - a 94.8% reduction from baseline. No staffing changes. No shift restructuring. No additional operational cost beyond the information layer itself. Critically, the system did not improve linearly across these conditions - it crossed a threshold. Below it the network remains structurally unstable regardless of partial information adoption. Above it, cognitive routing achieves load equilibrium and the Friction Transfer loop is broken at its origin point.

To find the cost-minimal configuration under high-adoption conditions I ran a 500-iteration constrained optimisation with hard wellbeing constraints - any configuration producing average frustration above 0.3 or average staff burnout above 0.3 was discarded as infeasible. The optimiser converged on its best feasible solution at iteration 173 at a total objective value of £5,778 per shift. The objective function penalises each SLA breach at £50, meaning the optimised configuration achieves near-elimination of service failures while minimising total operational cost. Staffing expansion at maximum physical capacity under partial adoption cannot compensate for informational failure. That is the core finding this project was designed to produce, and the experimental architecture I built was constructed specifically to be able to prove it.

{#2} The Baseline

The University of Nottingham Student Service Centre operated across two campuses - University Park and Jubilee. Under a 50 agents per hour arrival rate with a 60/40 geographic demand bias weighted toward Jubilee, the system produced a mean of 140.07 SLA breaches per shift across 1000 replications, where a single breach constitutes an agent waiting beyond the 15-minute threshold. One campus was overwhelmed. The other was underloaded. Students were waiting beyond the 15-minute threshold in sufficient numbers to constitute a systemic failure, not an occasional outlier. The standard deviation of 39.57 across 1000 replications confirms that the failure was not an artefact of a single unlucky run - it was a structural condition that manifested consistently with significant severity across the entire replication set. On paper the network had people, it had desks, it had capacity. In practice it was collapsing under regular operating conditions.

The obvious response to a network producing 140 breaches per shift is to hire more staff. More desks, more processing capacity, lower wait times. That logic is coherent given only the surface data. But adding staff to a system whose failure is driven by where demand lands, not how much capacity exists to serve it, does not change the outcome. It changes the cost. The network already had sufficient aggregate capacity across both campuses. The problem was that the people were in the wrong place at the wrong time, repeatedly, by design - because nothing in the system was directing demand toward available capacity. More staff allocated by the same flawed logic would have produced the same imbalance at higher operational cost.

Under the baseline condition - zero app adoption, every agent routing by proximity alone - agents had no visibility of queue state at either campus. Agents spawned at positions drawn from a 60/40 geographic distribution and routed to whichever node was physically closest using straight-line Euclidean distance. No queue length. No staff availability. No task-type compatibility. Distance only. Every student making that decision independently, under the same information vacuum, produced the same aggregate result - demand concentrated at the nearest node regardless of its current load, while available capacity at the further node went unused. The failure was produced by individually rational decisions made without the information that would have made them collectively rational.

The correct diagnosis changed the design question entirely. If the failure originates at the routing decision - made by an individual agent, before they arrive, with no information about system state - then the intervention point is not the service desk. It is the cognitive moment that precedes it. Designing for that moment meant designing a system that could surface the right information, to the right agent, in a form they could actually use given their digital literacy, their anxiety level, their social context, and the specific task they needed to complete. Not a notification. Not a sign. A decision support layer built around how humans actually process information under uncertainty - which is imperfectly, selectively, and with systematic bias toward the familiar. That is what this project was designed to build, and what the experimental architecture was constructed to validate.

{#3} Architecture Design Rationale

The architecture was not chosen for completeness. It was chosen because the question this project was designed to answer made two simpler architectures insufficient. A pure Discrete Event Simulation would have modelled the operational layer with accuracy - queue throughput, staff processing rates, SLA breach counts, resource utilisation across three skill tiers. But a DES treats agents as passive arrivals. They enter the system, join a queue, get served, and leave. The model has no mechanism for asking why they arrived at that campus rather than the other one, what information they had when making that decision, or what emotional state they were carrying when they reached the desk. A DES would have told me how the service node was performing. It could not have told me why the queue was forming there in the first place. That is not the question this project needed to answer.

A pure Agent-Based Model would have resolved that limitation. Individual agents making routing decisions based on perceived queue state, digital literacy, anxiety, social context, and campus reputation - that is exactly what the ABM layer does. But without a validated operational layer underneath, the SLA breach counts those agents produce are not credible. You cannot prove that a routing intervention works at a system level if the service environment itself is not modelled with fidelity. The ABM alone would have produced plausible individual behaviour with unverifiable system outcomes.

The hybrid was the only architecture that could answer the actual question. The DES layer governed the operational environment - queue logic, three independent staff resource pools across General, Specific, and Translator tiers, service time distributions modified dynamically by task type and linguistic requirements, and SLA breach auditing with per-campus counters feeding directly into the optimisation objective function. The ABM layer governed individual student cognition — agent attributes spanning identity, spatial position, psychological state, social context, information access, and campus reputation memory. Each layer was doing the work the other structurally could not.

Two mechanisms inside the ABM layer illustrate precisely why a DES alone was architecturally insufficient, and why agents needed to be modelled as spatially aware, cognitively distinct individuals rather than identical decision-makers processing the same inputs. The first is the predictive pressure engine, built on task-isolated shadow queues. When any agent - app user or blind user - commits to a campus and begins travelling, their inbound presence is registered through passive infrastructure detection. In the real-world system this assumption is underpinned by WiFi MAC address tracking: as a student begins moving toward a campus, their device is detected by the network and their inbound presence is logged before they physically arrive. In the simulation this is abstracted as an immediate increment to the correct task-specific shadow counter - `upSpecShadowQueue` , `upGenShadowQueue` , `jubSpecShadowQueue` or `jubGenShadowQueue` - at the moment of routing commitment. Every subsequent app user making a routing decision reads `predictedLoad = realQueueSize + shadowQueue`, meaning they are routing based on the anticipated future state of the node, not its current state. The shadow queue does not track why agents are coming or how they made their decision. It tracks aggregate inbound pressure regardless of routing method. Furthermore, the shadow queues are task-isolated by design - an agent needing Visa or complex task assistance reads only the specific task shadow counter and ignores general queue pressure entirely. This was a deliberate information architecture decision: showing an agent the wrong queue signal is worse than showing them nothing, because it produces false routing behaviour that degrades system performance rather than improving it. A pure DES has no mechanism for any of this. Agents in a DES do not exist until they enter the system. The shadow queue gives spatially distributed agents temporal awareness of a node they have not yet reached - something the DES layer structurally cannot provide.

The second mechanism is information asymmetry in app data. App users do not receive accurate queue data. Each agent computes an `uncertainityRange` as `(1 - (digitalLiteracy/100 * infoTrustScore)) * 0.3` , and independent noise values are generated separately for each campus within that range using `uniform(-uncertainityRange , uncertainityRange)` . An agent with low digital literacy or low trust in the system sees a significantly distorted picture of queue state. Because the noise is generated independently for each campus, the same agent can simultaneously overestimate University Park and underestimate Jubilee - or vice versa - producing systematically biased routing decisions that do not reflect ground truth. Two agents standing in identical spatial positions with different literacy levels will make different routing decisions from the same underlying queue data. This is not a behavioural detail added for realism. It is a core architectural argument for why agents had to be modelled as cognitively distinct individuals. A model that gave all app users identical, accurate queue information would have produced a routing optimiser, not a human decision support system. The noise mechanic is what makes the simulation honest about what information-guided routing actually looks like when deployed to a real population.

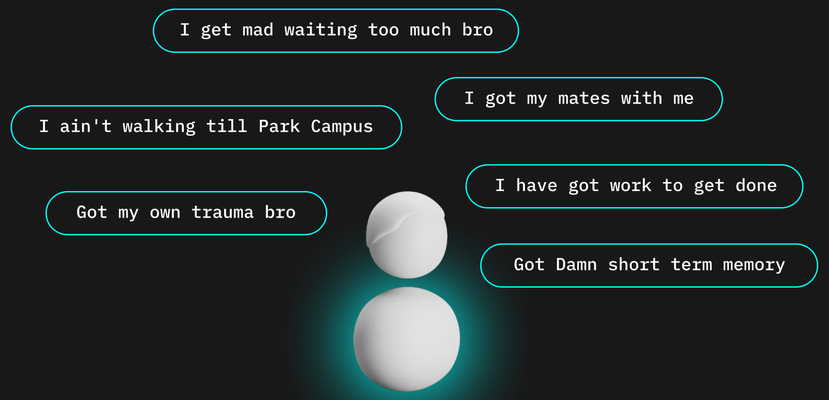
The variable that connects the two layers across the entire model is `rageIndex` - a continuous psychological friction score carried by each agent, bounded between 0 and 1, that accumulates across two distinct stages before it crosses from the cognitive layer into the operational layer. Understanding how it builds, and what it does when it arrives at the service node, is what makes the Friction Transfer mechanism precise rather than conceptual.

The first stage of rageIndex accumulation occurs inside `calculateOptimalRoute()` . For app users, the function computes a composite routing cost for each campus factoring in perceived queue pressure, cognitive bias scaled by current anxiety, active patience modified by social context, skill multipliers reflecting campus efficiency differences, transit distance amplified by a live weather multiplier, and a reputation penalty carried from the Peer's last campus visit. If the lowest available routing cost exceeds the agent's panic threshold - calculated as `activePatience * 5.0` - the thinking loop fires. RageIndex is set proportional to how far that threshold was breached, capped at 1.0. Anxiety then increments by `rageIndex * 0.2` , cognitiveBias is recalculated as `1.0 + currentAnxiety` , and the entire cost function runs again within the same decision cycle under the heightened cognitive state. The agent arrives at their chosen campus already carrying a rageIndex reflecting the psychological cost of the routing decision itself - before they have waited a single minute.

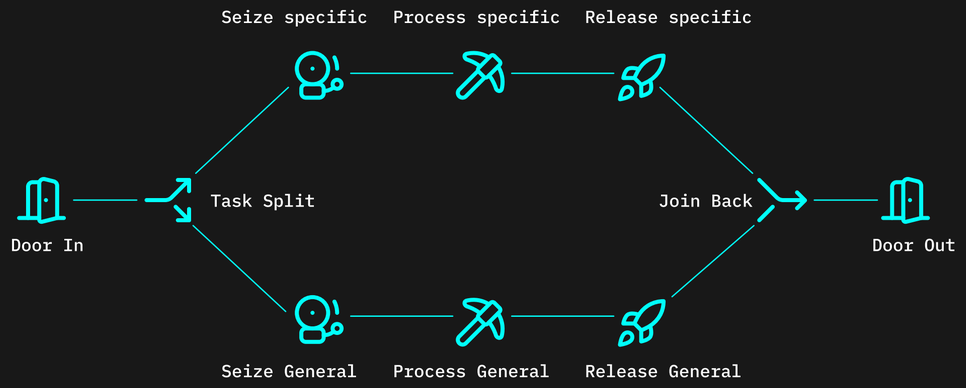
The second stage occurs at the seize block. If actual wait time exceeds 15 minutes, rageIndex escalates further by `breachSeverity * 0.05` for every minute beyond the threshold, capped at 1.0. The two stages are additive. When combined rageIndex reaches or exceeds 0.1, the campus node absorbs `rageIndex * 0.1` into its `staffFatigue` state, hard-capped at 1.0. That fatigue enters the process block as a live speed multiplier - `1.0 + staffFatigue` - meaning a node at maximum fatigue processes every subsequent agent at half its baseline speed. A `Staff_Recovery` event manages decay between service interactions. Under sustained hostile load, frustrated agents arrive faster than the recovery event can discharge and fatigue climbs toward its ceiling. The loop closes completely: frustrated agents slow service, slower service produces longer waits, longer waits produce more frustrated agents, those agents transfer more fatigue, service slows further. Modelling this was not optional. The reduction from 140.07 to 7.312 mean breaches is not simply a consequence of better routing. It is a consequence of breaking this loop at its origin point - the routing decision - before the frustration cascade begins. Without the Friction Transfer mechanism connecting the two layers, that finding would have been invisible.

{#4} Conceptual Model Diagram

Agent Character & Attributes



Discrete Event Attributes



{#5} Model Scope and Level of Detail

Agent Attributes

Identity and Task

`serviceType` • String, assigned at spawn as "general", "Visa", or "complex task".

`isAppUser` • boolean, determines whether the agent hits the Blindness Wall or uses the full cognitive routing engine.

`translationNeeded` • boolean, triggers compound resource constraints (Pool_Specific + Pool_Translator) and adds a service time penalty.

Spatial Context

`HomeX` • double, spawn X coordinate (biased 60/40), used for Euclidean distance calculations.

`HomeY` • double, spawn Y coordinate, used for Euclidean distance calculations.

Psychological State

`currentAnxiety` • double, initialises the cognitive bias multiplier, increments during the thinking loop.

`rageIndex` • double, bounded 0-1, accumulates during routing panic and queue breaches, drives the Friction Transfer to staff.

`waitSensitivity` • double, baseline patience score before social distraction is applied.

Social Context

`groupSize` • double, drives both the social buffering (cost reduction) and social distraction (patience reduction) calculations.

`peerInfluence` • double, scales the distraction coefficient, causing proximity reversion in group agents.

Information and trust

`digitalLiteracy` • double, determines the ceiling of the uncertainty range.

`infoTrustScore` • double, combined with digital literacy to compute independent noise bounds for queue perception.

Memory and Reputation

`lastCampusVisited` • String, records which campus was visited on the previous run.

`lastExperiencePenalty` • double, additive routing cost applied if the previous visit resulted in an SLA breach.

Operational State

`chosenNode` • String, locked at spawn, used for physical canvas routing and shadow queue decrements.

`isWaiting` • boolean, activates on entering the Seize block and deactivates on seizing a unit, enabling the rage accumulation flag.

`entryTime` • double, set at TimeMeasure_Start, the anchor point for all local waiting calculations.

`timeOfArrival` • double, set at TimeMeasure_Start, used exclusively in TimeMeasure_End for global diagnostics.

`pureWaitTime` • double, frozen at Seize exit, the official metric evaluated in the SLA audit.

⚡ DES Architecture Attributes

Parameters - Static Configuration

`nodeName` • String, identifies the campus node, used in diagnostic console logging throughout the service flow.

`slaThreshold` • double, the strict 15-minute threshold evaluated against pureWaitTime at the end of the pipeline.

Variables - Dynamic Operational State

`staffFatigue` • double, bounded 0-1. The core Friction Transfer accumulator. Receives rageIndex * 0.1 from enraged agents and acts as a multiplier on service delay times.

`currentQueueLength` • double, updated dynamically at the Task_Router to track real-time aggregate physical queue occupancy.

Resource Pools

`Pool_General` • resource pool, claimed by Seize_General as the primary option, and acts as the fallback from Pool_Specific due to one-way fungibility.

`Pool_Specific` • resource pool, claimed exclusively by Seize_Specific.

`Pool_Translator` • resource pool, claimed simultaneously with a primary pool when agent.translationNeeded is true.

Events

`Staff_Recovery` • timed event, manages the active decay of staffFatigue between service interactions to prevent permanent node collapse.

Global System & Optimisation Attributes

Predictive Shadow Queue Variables

`upSpecShadowQueue` • String, locked at spawn, used for physical canvas routing and shadow queue decrements.

`upGenShadowQueue` • integer, tracks UP general inbound pressure.

`jubSpecShadowQueue` • integer, tracks Jubilee specific inbound pressure.

`jubGenShadowQueue` • integer, tracks Jubilee general inbound pressure.

SLA Audit Counters

`clean_Total_Breaches` • integer, incremented at TimeMeasure_End across both campuses.

`clean_UP_Breaches` • integer, incremented when an SLA breach is attributed to University Park.

`clean_Jub_Breaches` • integer, incremented when an SLA breach is attributed to Jubilee.

Staffing Optimisation Decision Variables

`upGeneralStaff` • integer, sets Pool_General capacity at UP.

`upSpecificStaff` • integer, sets Pool_Specific capacity at UP.

`upTranslator` • integer, sets Pool_Translator capacity at UP.

`jubGeneralStaff` • integer, sets Pool_General capacity at Jubilee.

`jubSpecificStaff` • integer, sets Pool_Specific capacity at Jubilee.

`jubTranslator` • integer, sets Pool_Translator capacity at Jubilee.

Objective Function Wage Parameters

`wageGeneral` • double (£15/hr), fixed multiplier for optimisation constraints.

`wageSpecific` • double (£25/hr), fixed multiplier for optimisation constraints.

`wageTranslator` • double (£20/hr), fixed multiplier for optimisation constraints.

Routing and Environmental Modifiers

`AppUserSlider` • UI control, dynamically sets the probability of an agent routing via the cognitive engine vs. the Blindness Wall.

`upX, upY` • double, spatial coordinates of the UP node.

`jubX, jubY` • double, spatial coordinates of the Jubilee node.

`pressureThreshold` • double, load threshold that triggers the system to apply a balancing penalty.

`balancingBias` • double, dynamic penalty applied to actively steer agents away from critically overloaded campuses.

`queueWeightingFactor` • double, globally scales the influence of predicted queue load in the perceived cost calculation.

`weatherMultiplier` • double, live environmental friction variable that amplifies transit distance costs, triggering stochastic black swan events.

{#6} The Behavioural Model - 5 Mechanisms

The cognitive layer of this model is not a routing algorithm. It is a simulation of how humans actually make decisions under uncertainty - with incomplete information, emotional state, social pressure, and accumulated experience all shaping the output. Every mechanism inside `caluclateOptimalRoute` represents a deliberate design decision grounded in a specific behavioral rationale. This section explains each one in the order they execute.

Mechanism 1 - The Blindness Wall

The first operation `calculateOptimalRoute()` performs is a binary check: `if (this.isAppUser == false)` . If the agent does not have the app, the function executes a fundamentally different and deliberately impoverished decision process. It calculates straight-line Euclidean distance to both campuses using the agent's spawn coordinates `Math.sqrt(Math.pow(root.upX - homeX, 2) + Math.pow(root.upY - homeY, 2))` - compares the two values, selects the smaller, increments the correct specific or general shadow queue, and returns. The function exits at this point. No queue data is read. No staff availability is checked. No task-type compatibility is evaluated. Distance is the entire decision. This was not a simplification. It was the most honest model of how a student without real-time information actually behaves. In the absence of any other signal, proximity is the only rational heuristic available. The Blindness Wall captures the baseline cognitive condition the information layer was designed to overcome - not irrational behaviour, but rational behaviour operating on structurally inadequate information. The 60/40 geographic demand bias means that 60% of blind users spawn in the Jubilee-proximate region and default to Jubilee regardless of its queue state. This is the structural mechanism that produces the 140.07 mean breach baseline - not insufficient total capacity, but systematically misdirected demand.

Mechanism 2 - Bounded Information

For app users, the function does not grant perfect visibility of queue state. It immediately computes the limits of each agent's informational accuracy: `uncertaintyRange = (1.0 - ((digitalLiteracy / 100.0) * infoTrustScore)) * 0.3` . This value represents the maximum proportional distortion the agent will apply to queue data. An agent with perfect digital literacy and full trust in the system has an uncertaintyRange of zero - they see ground truth. An agent with low literacy and low trust has an uncertaintyRange approaching 0.3 - their perception of the queue can be off by up to thirty percent in either direction.

Two independent noise values are then generated - `upNoise = 1.0 + uniform(-uncertaintyRange, uncertaintyRange)` and `jubNoise` separately - meaning the distortions for each campus are statistically independent. The same agent can simultaneously overestimate University Park and underestimate Jubilee, or underestimate both, or any combination within their uncertainty bounds. This independence matters because it prevents the model from behaving as a symmetric error correction system

Real perceptual errors are not symmetric. Two agents standing in the same position with different digital literacy profiles will read the same queue and make different routing decisions. That is what this mechanism models.

The design decision here was to prioritise honesty over optimism. A model that gave all app users accurate queue data would have validated the best-case scenario for the information layer. Building in literacy-dependent noise validated a realistic deployment scenario - one where the population using the system has varying capacity to interpret it correctly.

Mechanism 3 - Asymmetric Social Gravity

When an agent is travelling in a group, two competing social forces act on their decision simultaneously. The first is social buffering: . `socialBuffering = 1.0 + (Math.log (groupSize + 1) * 0.25)` This value divides the perceived queue load for both campuses in the cost calculation, making both destinations appear less psychologically costly. Companionship softens the perceived burden of any queue regardless of which campus the agent is considering. The second is social distraction: `activePatience = Math.max(5.0, waitSensitivity - (peerInfluence * (Math.log(groupSize + 1) * 0.5)))` This subtracts from the agent's patience value, producing a lower `activePatience` score for agents travelling in larger groups. Because `activePatience` appears as a multiplicative factor in the composite cost function - `cost = (perceivedLoad * cognitive Bias * activePatience * skillMultiplier) + transitDistance` - a lower value reduces the computed cost of queue pressure at both campuses proportionally. The critical structural consequence of this is that transit distance, which is additive rather than multiplicative, becomes relatively more influential in the agent's decision as `activePatience` decreases active patience An agent with low `activePatience` assigns less absolute weight to queue data and more relative weight to proximity. They behave more like a proximity-defaulting agent - closer is cheaper - even when they have access to queue information.

This models a coherent behavioural phenomenon: an agent under social pressure from companions is less willing to engage analytically with queue data and more likely to fall back on the simpler proximity heuristic. The social distraction is not making them more sensitive to long queues - it is making them less sensitive to queue information altogether and more responsive to the immediate spatial signal of distance. The practical result is that heavily socially-distracted app users partially converge toward the behaviour of blind users, defaulting to proximity even when queue data suggests a better option exists. The coefficients are deliberately asymmetric. Buffering uses a multiplier of 0.25. Distraction uses a multiplier of 0.5. The distraction force is twice as strong as the buffering force - reflecting that social obligation erodes analytical decision-making more strongly than companionship reduces the psychological weight of waiting.

Two additional constraints govern this mechanism. The logarithmic scaling - `Math.log(groupSize + 1)` - means social effects grow with group size but with diminishing returns. A group of ten does not produce ten times the distraction of a group of one. And `activePatience` is hard-floored at 5.0, preventing social pressure from reducing the patience value to zero or below, which would have collapsed the cost function and produced pathological routing behaviour in large groups.

Mechanism 4 - The Composite Cost Function

Once social variables are computed, the function builds a composite routing cost for each campus across four components.

The first component is perceived queue pressure. The function reads the real current queue size for the agent's task type - `realUP = isSpecificTask ? root.node_UP.Seize_Specific.queue.size() : root.node_UP.Seize_General.queue.size()` - using `.queue.Size()` to access the actual waiting agents within the seize block rather than total block occupancy. It adds the relevant shadow queue to produce a predicted load: `predictedUP = realUP + relevantUpShadow` . That predicted load is then normalised against available staff: `upPressure = predictedUP / Math.max(1, staffUP)`. If pressure at one campus exceeds `pressureThreshold` and is higher than the other campus, a `balancingBias` multiplier is applied to that campus's perceived load - a dynamic steering mechanism that discourages agents from routing toward already-overloaded nodes. The final perceived load is: `perceivedUP = (predictedUP * upNoise * root.queueWeightingFactor * upSLAFactor) / socialBuffering` Social buffering divides at this stage, softening the perceived weight of the queue for agents travelling in groups.

The second component is campus skill efficiency. University Park applies a 0.9 multiplier for specific task agents - it processes them faster. Jubilee applies 1.1 - it processes them slower. For translation requirements, both campuses apply a 0.9 multiplier reflecting the processing shift when linguistic support is required: `upSkill = (isSpecificTask ? 0.9 : 1.0) * (translationNeeded ? 0.9 : 1.0)` . An agent needing Visa assistance with translation carries a compound skill multiplier that makes the Jubilee cost significantly higher than a general query agent would compute.

The third component is transit distance, computed as Euclidean distance from spawn coordinates and scaled by two factors: a 0.001 normalisation constant preventing raw pixel distances from overwhelming queue weights in the cost equation, and a live `weatherMultiplier` that amplifies transit friction under adverse conditions. `distUP = Math.sqrt(Math.pow(root.upX - homeX, 2) + Math.pow(root.upY - homeY, 2)) * 0.001 * root.weatherMultiplier`. Under severe weather, what was a negligible distance cost becomes a meaningful routing deterrent. Because distance is additive while queue pressure is multiplicative, weather events and reduced `activePatience` both shift the balance of the cost function toward proximity sensitivity - the two effects compound under simultaneous severe weather and high social distraction, which is part of the mechanism that produces the black swan collapse events visible in the σ=12.68 at 80% adoption.

The fourth component is campus reputation memory. If the agent or peers have previously visited a campus and that visit resulted in a poor experience, a `lastExperiencePenalty` is added to that campus's cost through two agent attributes - `lastCampusVisited` and `lastExperiencePenalty` - that persist across routing decisions. This abstracts word-of-mouth social diffusion: agents do not return to campuses that previously failed them, and that avoidance behaviour accumulates across the simulation.

The final composite cost for each campus is: `cost = (perceivedLoad * cognitiveBias * activePatience * skillMultiplier) + transitDistance + reputationPenalty` . The campus with the lower total cost is selected as the preliminary routing choice.

Mechanism 5 - The Thinking Loop

Before the decision is finalised, the function evaluates whether the lowest available cost exceeds what the agent can psychologically tolerate. The panic threshold is `rageTrigger = activePatience * 5.0` . If `chosenCost > rageTrigger` , the thinking loop fires.

`rageIndex` is computed as `Math.min(1.0, (chosenCost - rageTrigger) / (rageTrigger * 2.0))` - proportional to how far the cost exceeded the threshold, capped at 1.0. Anxiety then increments by `rageIndex * 0.2` , and `cognitiveBias` is recalculated as `1.0 + currentAnxiety` . The entire cost function then runs again - same perceived loads, same distances, same skill multipliers, same reputation penalties- but now under an elevated cognitive bias that amplifies the perceived weight of every variable in the multiplicative portion of the equation. The agent reconsiders their routing decision under a heightened stress state within the same decision cycle.

The design rationale for this mechanism is precise. An agent whose best available option still exceeds their tolerance does not simply accept it calmly. They panic. They recalculate under emotional amplification. Because `cognitiveBias` multiplies the perceived load component, a panicking agent computes higher absolute costs for both campuses - but because the relative ordering of costs is preserved in most cases, the recalculation primarily functions as a mechanism for shifting the agent toward the lower-pressure campus when the cost differential is meaningful. The thinking loop models stress-induced recalibration, where anxiety does not paralyse decision-making but amplifies sensitivity to the differences between available options.

{#7} The Service Architecture - DES Procedural Flow

The Discrete Event Simulation layer is not a passive backdrop to the cognitive routing engine. It is the operational environment that every agent passes through after their routing decision is made, and it is where the consequences of that decision become measurable. Every block in the service pipeline has logic running through it. Every transition between blocks captures or transforms something meaningful about the agent's state or the campus's operational condition. This section documents the full procedural flow from campus entry to exit, explaining what each block does, what code executes inside it, and why each design decision was made the way it was.

The campus service node carries parameters and variables governing its operational state. The parameters `nodeName` , `slaThreshold` , `ambientNoiseBaseline` `specificStaffCount` , `capacityLimit` , `baseTransit` , `generalStaffCount` , `translatorCount` - define the static configuration of the node. The dynamic variables - `activeStaffcount` , `stressMultiplier` , `staffFatigue` , `totalSLA_Breaches` , `currentQueueLength` , `translatorUsage` - track its live operational state throughout the simulation. Three resource pools - `Pool_General` , `Pool_Specific` , and `Pool_Translator` - represent the discrete human capacity available to process demand. A `Staff_Recovery` event manages fatigue decay between service interactions.

Source Block and Select_Campus

Before entering any service node, every agent passes through two critical setup stages on the main canvas.

The source block assigns all immutable identity attributes at spawn in a precise sequence. The geographic position is assigned using a 60/40 split - `randomTrue(0.60)` places agents in the Jubilee-proximate region with coordinates `homeX = uniform(700, 1000)` , `homeY = uniform(600, 1000)` , while the remaining 40% spawn in the UP-proximate region with `homeX = uniform(0, 300)` , `homeY = uniform(0, 400)` . This spatial bias reflects the accommodation density reality of the campus network and is the foundational condition that makes the routing research question meaningful - without geographic asymmetry, proximity routing accidentally balances load and the information layer has nothing to fix.

The source block assigns all immutable identity attributes at spawn in a precise sequence. The geographic position is assigned using a 60/40 split - `randomTrue(0.60)` places agents in the Jubilee-proximate region with coordinates `homeX = uniform(700, 1000)` , `homeY = uniform(600, 1000)` , while the remaining 40% spawn in the UP-proximate region with `homeX = uniform(0, 300)` , `homeY = uniform(0, 400)` . This spatial bias reflects the accommodation density reality of the campus network and is the foundational condition that makes the routing research question meaningful - without geographic asymmetry, proximity routing accidentally balances load and the information layer has nothing to fix.

App status is assigned immediately after: `agent.isAppUser = randomTrue(AppUserSlider.getValue())` . Service identity is then assigned through a two-stage random split: `agent.serviceType = randomTrue(0.5) ? "general" : (randomTrue(0.4) ? "Visa" : "complex task")` This produces a population of approximately 50% general task agents, 20% Visa agents, and 30% complex task agents. Visa and complex task agents combined constitute the 50% who will enter the specific service pathway. This assignment is permanent - the agent carries this identity for their entire journey through the model.

Finally, `agent.chosenNode = agent.calculateOptimalRoute()` locks the campus decision before the agent begins physical movement. This prevents any race condition between the cognitive decision and the physical routing mechanism.

From spawn, agents travel physically across the main canvas environment toward their chosen campus. A `Select_Campus` block serves as the physical routing junction, evaluating `agent.chosenNode.equals("UP")` to direct the agent to the correct service node. This is the mechanism that translates the cognitive routing decision into actual movement - the string returned by `calculateOptimalRoute()` becomes the physical path the agent takes.

Door_In

The agent enters the campus service node through Door_In. No code executes here. Door_In is a structural entry point - its sole function is to receive the agent from the spatial environment and pass them into the service pipeline.

TimeMeasure_Start

TimeMeasure_Start is the first point of operational logic and it performs four distinct operations simultaneously.

The first two record time. `agent.timeOfArrival = time()` and `agent.timeOfArrival = time()` are set identically at this moment, establishing the reference timestamp that every subsequent time-dependent calculation in the pipeline will measure against. These are not redundant - `timeOfArrival` I tracks campus arrival for longitudinal analysis in the TimeMeasure_End diagnostic, while `entryTime` is the active measurement variable used by the seize block and process block to compute wait duration.

The third operation is the shadow queue decrement - what the code comments call "The Honest Math Handshake." When the agent committed to this campus during `calculateOptimalRoute()`, they incremented the correct task-specific shadow queue counter. The moment they physically arrive at TimeMeasure_Start, that predictive registration is removed. The decrement uses `agent.chosenNode` - the agent's own recorded routing decision - rather than inferring campus identity from the block's location: `if (agent.chosenNode.equals("UP")) { if (isSpec) main.upSpecShadowQueue--; else main.upGenShadowQueue--; }` This is architecturally more precise than checking physical block location. Without this decrement, every arriving agent would be counted simultaneously as inbound shadow pressure and actual queue load, causing the predictive signal to inflate progressively and producing phantom congestion that would corrupt routing decisions across the entire simulation run.

The fourth operation is a safety floor: all four shadow queue counters are checked against zero and reset if they go negative, preventing arithmetic errors from corrupting the predictive signal under edge-case timing conditions.

Task_Router

From TimeMeasure_Start the agent moves to Task_Router, the bifurcation point of the service pipeline. The router evaluates a condition - `agent.serviceType.equals("complex task") || agent.serviceType.equals("Visa")` - consistent with the `isSpecificTask` boolean defined throughout the routing engine, and simultaneously updates the campus's live queue length: `currentQueueLength = Seize_Specific.size() + Seize_General.size()`.

The routing split is binary and consequential. Agents with serviceType "complex task" or "Visa" - 50% of the population by spawn configuration - route to the Specific pathway. Agents with serviceType "general" route to the General pathway. This split determines which resource pool the agent will compete for, which service time distribution will govern their processing, and therefore how long they will wait and whether they will breach the SLA threshold.

Seize_Specific

The Seize_Specific block contains logic across four distinct execution points - resource sets, on enter, on seize unit, and on exit - and it is where the Friction Transfer mechanism crosses from the cognitive layer into the operational layer.

The resource set definition governs which staff the agent can claim. If the agent requires translation, the block requests a simultaneous resource pair: `{Pool_Specific, Pool_Translator}`. If no translation is needed, it requests `{Pool_Specific}` alone. This means an agent needing linguistic assistance cannot be served until both a specific staff member and a translator are simultaneously available - a compound resource constraint that accurately reflects the operational reality of complex multilingual casework

On enter, `agent.isWaiting = true` activates the rage accumulation flag.

On seize unit - the moment a staff resource is claimed - five operations execute in sequence. First, `agent.isWaiting = false` deactivates the rage accumulation flag. Second, actual wait time is calculated: `actualWait = time() - agent.entryTime`. Third, if `actualWait > 15.0`, rageIndex escalates by `breachSeverity * 0.05` for every minute beyond the threshold, capped at 1.0. An agent who waited 20 minutes arrives at the desk with a rageIndex increment of 0.25 regardless of their initial psychological state. Fourth, if combined rageIndex is at or above 0.1, the Friction Transfer fires: `this.staffFatigue += (agent.rageIndex * 0.1)`, hard-capped at 1.0. Fifth, diagnostic logging records the transfer severity - agents with rageIndex above 0.5 trigger a system warning, those between 0.1 and 0.5 trigger a system note, and calm agents produce a clean seize confirmation.

On exit, `agent.pureWaitTime = time() - agent.entryTime` freezes the queue wait measurement at the exact moment the agent transitions from waiting to being served. This is the value the SLA audit will check at TimeMeasure_End - critically distinct from `actualWait`. Both measure elapsed time from entryTime, but they are frozen at different moments. `actualWait` is captured during the seize unit event and drives the psychological and Friction Transfer mechanics. `pureWaitTIme` is frozen at seize exit and constitutes the official SLA measurement.

Seize_General

Seize_General follows the same structural logic as Seize_Specific with one architecturally significant difference in its resource set definition. Where Seize_Specific offers only `{Pool_Specific}` Seize_General offers two alternatives: `{Pool_Specific}` or `{Pool_General}` - and their translation variants. This is structural one-way fungibility. Specialist staff can step down to absorb general demand when General pools are exhausted. General staff cannot step up to handle complex or Visa casework regardless of availability. This asymmetry is architecturally enforced in the resource set definitions rather than approximated through probability weights - the constraint is structural, not statistical.

The on enter, on seize unit, and on exit logic in Seize_General is identical to Seize_Specific. The same rage escalation, the same friction transfer threshold at 0.1, the same `pureWaitTime` freeze at exit.

Process_Specific

From Seize_Specific the agent enters Process_Specific. The on enter code re-stamps `agent.pureWaitTime = time() - agent.entryTime` - locking the queue wait duration before the service delay begins - and logs the current fatigue multiplier: `1.0 + this.staffFatigue`.

The delay time formula is: `((agent.serviceType.equals("Visa") ? uniform(30, 60) : uniform(10, 20)) + (agent.translationNeeded ? uniform(2, 8) : 0.0)) * (1.0 + this.staffFatigue)`

This formula has three components. The base service time differentiates by task type within the specific pathway - Visa cases take between 30 and 60 minutes reflecting genuine visa processing complexity, complex non-Visa tasks take between 10 and 20 minutes. If translation is required, an additional 2 to 8 minutes is added. The entire sum is then multiplied by `1.0 + this.staffFatigue`. At zero fatigue the multiplier is 1.0. At the cap of 1.0 every interaction takes twice as long. The multiplier compounds across both the task time and the translation overhead simultaneously - a fatigued node processing a Visa case with translation is amplifying an already-elevated base duration.

Process_General

Process_General follows the same structure with a simpler base time distribution. The delay formula is: `(uniform(10, 20) + (agent.translationNeeded ? uniform(2, 8) : 0.0)) * (1.0 + this.staffFatigue)`. General tasks draw from a 10 to 20 minute base distribution. The translation penalty and fatigue multiplier apply identically. The distinction between the two pathways is resource pool allocation and service time complexity - not the fatigue mechanic, which operates identically across both.

Release_Specific and Release_General

No code executes in either Release block. Their function is structural - they return the claimed resource units back to their respective pools immediately, making capacity available for the next agent in the queue.

TimeMeasure_End

TimeMeasure_End is where the SLA audit executes. The block performs a diagnostic comparison between `agent.pureWaitTime` and `time() - agent.timeOfArrival` to verify measurement consistency, then evaluates `agent.pureWaitTime` against `slaThreshold`. If `pureWaitTime > slaThreshold`, three counters increment simultaneously: `main.clean_UP_Breaches++` updates the global system breach count, and either `main.clean_UP_Breaches++` or `main.clean_Jub_Breaches++` updates the campus-specific counter using `agent.chosenNode` - the agent's own recorded routing decision. These per-campus counters feed directly into the optimisation objective function. Each breach carries a £50 penalty weight. The audit measures pureWaitTime exclusively - not actualWait, not total time in system - because the SLA definition is a queue wait threshold.

Door_out

The agent exits through Door_Out. No code executes here. The pipeline is complete.

Four Architectural Decisions that Defines the Model’s Integrity

The pipeline contains four design decisions not visible in the block diagram but essential to the model's validity.

The first is the shadow queue handshake at TimeMeasure_Start. The decrement keeps the predictive signal and actual queue load from simultaneously counting the same agent, preventing phantom congestion from accumulating across the simulation run.

The second is the use of `agent.chosenNode` for campus identification in both TimeMeasure_Start and TimeMeasure_End. Rather than inferring campus identity from which physical node the block belongs to, both blocks read the agent's own recorded decision. This makes the campus attribution system architecturally precise and robust to edge-case ambiguity.

The third is one-way fungibility in the resource sets. Encoding specialist absorption of general demand as a structural constraint means the model cannot route a general agent through a specific staff member unless Pool_General is genuinely exhausted. The constraint is architectural, not statistical.

The fourth is the dual time measurement - `actualWait` for psychological and fatigue calculations, `pureWaitTime` for the SLA audit. These measure two genuinely different durations that share a start point. `actualWait` captures the psychological experience of waiting and drives the Friction Transfer mechanism. `pureWaitTime` captures the operationally auditable queue duration and drives the breach counter. Conflating them would have produced either an unfair SLA metric or an inaccurate frustration model.

{#8} Experimental Design & Hypothesis Testing

The three experimental phases were not configurations chosen to produce favourable results. They were hypothesis tests designed in sequence to isolate a specific causal claim - that informational architecture, not operational capacity, determines whether a service network succeeds or fails under load.

The geographic distribution decision. During model development, a 50/50 geographic spawn distribution was tested. Under that condition, results at 0% and 90% adoption were statistically indistinguishable because proximity routing accidentally balanced load when demand was symmetrically distributed. The research question - does information-guided routing fix structural load imbalance - requires structural load imbalance to exist in the first place. A 50/50 distribution creates a null problem. The 60/40 distribution, weighted toward Jubilee, creates the demand concentration condition that makes the routing intervention meaningful and the research question testable. This is not a post-hoc adjustment to produce better results. It is the correct experimental condition for the hypothesis being tested.

The critical experimental decision that made all three phases credible was holding staffing constant throughout. University Park carried the same staffing across General, Specific, and Translator tiers across all three phases. Jubilee carried the same. The only variable that changed was app adoption rate. If staffing had changed between phases, any improvement in breach counts could have been attributed to headcount rather than information compliance. Fixing staffing as a constant meant every result across all three phases had a single explanation: what changed was how students routed, and what changed routing was information availability.

The arrival rate was set at 50 agents per hour throughout - a deliberate stress condition. Each phase ran 1000 replications to produce statistically stable mean estimates and to reveal the full variance structure of the system's behaviour.

Phase 1 - Confirming the Failure Mode

Hypothesis : Proximity-only routing produces structural overload at one node despite sufficient aggregate capacity, manifesting as systemic SLA failure rather than marginal degradation.

Configuration : 0% app adoption. Every agent executed the Blindness Wall - pure Euclidean distance routing with no queue visibility.

Result : Mean SLA breaches 140.07, standard deviation 39.57.

What it proved : the failure mode is real, severe, and structurally produced. The standard deviation of 39.57 across 1000 replications confirms consistent failure - not an outlier condition. The variance reflects the stochastic elements of spawn position, service time, and weather events, all of which modulate the severity of failure without changing its fundamental character. Phase 1 established the ground truth every subsequent phase measured against.

Phase 2 - Testing Partial Intervention

Hypothesis : Introducing real-time routing information for a subset of the population will reduce system failure, but partial adoption will be insufficient to stabilise the network because blind users will continue to concentrate demand at proximity-defaulting nodes.

Configuration : 50% app adoption. Half the agent population executed the full `calculateOptimalRoute()` cognitive engine. The other half remained on the Blindness Wall. Staffing unchanged.

Result : Mean SLA breaches 23.914, standard deviation 26.90. An 82.9% reduction from baseline.

What it proved : Partial information compliance produces substantial but incomplete recovery. The 82.9% reduction confirms the information layer works at scale. But the remaining 23.914 mean breaches and the standard deviation exceeding the mean confirm that the Friction Transfer loop is not broken at 50% adoption. Enough blind users continue routing by proximity to sustain overload events at Jubilee under adverse stochastic conditions. The intervention is working. It is not yet working at the scale required for systemic stability.

Phase 3 - Finding the Stability Threshold

Hypothesis : At sufficiently high adoption, cognitive routing achieves load equilibrium and the Friction Transfer loop is broken at its origin point, producing near-elimination of SLA failure without any operational restructuring.

Configuration : 80% app adoption. Four in five agents executed the full cognitive routing engine. One in five remained on the Blindness Wall. Staffing unchanged.

Result : Mean SLA breaches 7.312, standard deviation 12.68. A 94.8% reduction from baseline.

What it proved : The hypothesis was confirmed. Near-elimination of breaches without a single additional staff member, without any shift restructuring, without any operational intervention beyond the information layer. The residual 7.312 mean breaches represent two distinct populations - the 20% of blind users who default to proximity regardless of system state, and the occasional severe weather event that simultaneously compresses demand and routing options beyond the information layer's ability to compensate.

The Non-Linearity Finding

The most significant result across the three phases is the shape of the improvement curve. Between Phase 1 and Phase 2 the system improves substantially — 82.9% reduction. Between Phase 2 and Phase 3 the reduction is a further 12 percentage points in absolute terms but it represents the transition from an unstable system to an effectively stable one. This non-linearity has a direct operational implication: there is a minimum viable adoption rate below which the information layer produces partial benefit without systemic recovery. The experimental design was constructed to force that threshold into visibility, and Phases 2 and 3 together locate it between 50% and 80% adoption under the stress conditions tested.

The Variance Structure as a Finding

The standard deviation pattern across the three phases reveals a second finding. At 0% adoption σ=39.57 - large variance around a high mean, reflecting variability in how severely the proximity routing failure manifests. At 50% adoption σ=26.90 - variance remains high because the system is oscillating between partial recovery and collapse depending on stochastic conditions. At 80% adoption σ=12.68 - the mean is near zero but the standard deviation exceeds the mean, confirming a bimodal distribution. Most of the 1000 runs produced zero or near-zero breaches. A minority produced significantly higher counts. Those are the black swan weather events - environmental shocks that overwhelm the information layer precisely when it is most needed. This is the brittleness finding: the system is reliable under normal operating conditions and fragile under environmental shock. Optimisation for efficiency and design for resilience are different problems, and the experimental architecture was constructed to solve only the first one.

Phase 4 - The Optimisation

Objective : To find the cost-minimal staff configuration under high adoption conditions that minimises total operational cost - defined as the sum of staff wages and SLA breach penalties - subject to physical capacity constraints and hard human wellbeing constraints.

Objective Function f(x) : `root.calculateTotalCost()` - minimising the sum of staff wages across both campuses plus £50 per SLA breach recorded during the shift.

Wage Structure : General staff £15/hr, Specific staff £25/hr, Translator staff £20/hr.

Capacity Constraints : University Park capped at 25 total staff. Jubilee capped at 20 total staff.

Wellbeing Requirements : Any configuration producing average frustration above 0.3 or average staff burnout above 0.3 discarded as infeasible. These constraints ensure that cost minimisation cannot achieve its objective through staff exhaustion.

Configuration : 500 iterations, 100 replications per iteration. App adoption locked at 90% before each simulation run via `root.AppUserSlider.setValue(0.9)` OptQuest optimiser with random seed for unique simulation runs across iterations.

Optimal Configuration found at iteration 173:

Resources Tier	University Park	Jubilee
General Staff	2	5
Specific Staff	9	8
Translator	4	5
Total	15	18

Total Objective Function : **£5,778 per shift**

Wage Breakdown : University Park - (2×£15) + (9×£25) + (4×£20) = £335/hr × 8 hours = £2,680. Jubilee - (5×£15) + (8×£25) + (5×£20) = £375/hr × 8 hours = £3,000. Total wages per shift: £5,680. SLA penalty component: £5,778 - £5,680 = £98, equating to approximately 1.96 expected breaches per shift at £50 per breach - effectively zero service failure.

Objective : The optimiser concentrated Specific staff heavily at both campuses - 9 at University Park and 8 at Jubilee - reflecting the compound service demand created by Visa cases taking 30-60 minutes and complex tasks taking 10-20 minutes within the specific pathway. General staff allocation is comparatively lean at both nodes, consistent with the 50% general task population being served by the faster 10-20 minute base time. The Translator allocation of 4 at UP and 5 at Jubilee reflects the compound resource constraint - Visa and complex agents requiring translation cannot be served until both Specific and Translator staff are simultaneously available, making Translator under-allocation a bottleneck that the optimiser correctly penalises. The near-equal Specific allocation between UP and Jubilee - despite UP having a skill efficiency advantage - reflects the geographic demand reality: 60% of agents spawn near Jubilee and even with 90% app adoption a significant proportion routes there, requiring substantial specific capacity at that node.

{#9} The Brittleness Finding

The high standard deviation at 80% adoption - σ=12.68 against a mean of 7.312 - is not noise in the simulation. It is a precise quantitative characterisation of the system's failure mode under environmental shock, validated across 1000 replications.

Under normal operating conditions at 80% adoption, the cognitive routing engine distributes demand effectively. Most runs produce zero or near-zero breaches. The Friction Transfer loop never initialises because the sustained hostile load that drives its tightening never builds. Staff fatigue remains negligible. The `Staff_Recovery` event keeps pace with the minor accumulation from routing-anxiety transfers.

Under severe weather events - stochastically generated throughout the simulation - two things happen simultaneously. Transit friction increases via the `weatherMultiplier`, amplifying the distance component of every agent's routing cost. Simultaneously, the reduced weight of queue signal relative to transit distance - compounded by social distraction reducing `activePatience` in group agents - means some app users partially revert toward proximity-defaulting behaviour. These two effects combine to produce a brief but intense surge of demand at one node even among app users. That surge is sufficient to initiate the Friction Transfer cascade, producing the high-breach outlier runs that inflate the standard deviation.

This finding has a direct systems implication. The information layer that creates efficiency and stability under normal conditions also creates a specific vulnerability under environmental shock - it depends on agents perceiving a meaningful cost difference between campuses, and weather events compress that difference precisely when reliable routing is most critical. Resilience under shock requires a different design response than efficiency under normal load. The optimisation solved the efficiency problem. The resilience problem remains open and constitutes the most important direction for future work on this system.

{#10} Strategic Insights

Five explicit claims emerge from this project that generalise beyond the specific system modelled.

Information visibility is an operational infrastructure problem, not a communications problem

Providing students with a poster showing average queue times would not have produced these results. The intervention works because it surfaces real-time, task-specific, predictive queue state at the moment of routing decision - before the agent commits to a direction. The precision of the information architecture matters as much as its existence.

Human cognitive behaviour is a system variable, not a user experience variable

Digital literacy, anxiety, social context, and campus reputation are not soft factors to be considered after the system is designed. They are variables that determine where demand lands in a service network. Ignoring them produces a model that is technically accurate and operationally meaningless.

Adoption rate functions as a criticality lever, not a gradient

The improvement from 0% to 50% adoption is substantial. The improvement from 50% to 80% crosses a qualitative threshold - from unstable to stable. This means adoption strategy is not "more is better, incrementally." It is "identify the threshold and design an adoption programme specifically to cross it."

Staff wellbeing is a second-order output of information architecture quality

The reduction in burnout is not a direct consequence of lower workload. It is a consequence of fewer frustrated agents reaching the desk, which means less emotional labour transferred, which means fatigue accumulates slower than recovery can discharge. Improving information architecture is a staff wellbeing intervention. That reframe has implications for how service network investments are justified and evaluated.

Optimisation and resilience require separate design responses

The optimiser found the cost-minimal configuration for near-zero breach performance under normal conditions. It cannot solve the brittleness problem because brittleness emerges from stochastic environmental shocks that are outside the optimisation's control variables. Any organisation deploying this system needs to design for both - an efficiency layer for normal operations and a resilience layer for shock conditions. These are different problems requiring different interventions.

{#11} Limitations,Assumptions and Vulnerabilities

Static trust levels

The current model assumes that agent trust in the application (`infoTrustScore`) remains fixed throughout the simulation. The architecture assumes a user base with static memory; agents do not update their trust parameters based on whether previous app predictions were accurate. In reality, trust is highly volatile. A user who follows the app's guidance and still experiences a severe wait time will rapidly lose faith in the system, altering their future routing behavior. By holding this variable static, the model assumes artificial user compliance during repeated systemic failures.

Epistemology of Soft Variables - Quantification Trap

The cognitive routing engine successfully translates qualitative human states into programmable mathematical logic. However, this creates an inherent epistemic vulnerability: the over-quantification of soft variables: human emotion is fluid, subjective, and highly contextual, yet the simulation requires it to function as a precise mathematical parameter For example, the model defines a student's panic threshold precisely as `activePatience * 5.0` , and calculates social distraction using a specific logarithmic multiplier based on group size. While this produces mathematically coherent simulation data, empirically validating these exact coefficients in a physical setting is nearly impossible. The risk of this architecture is that the simulation's internal precision creates a false sense of real-world predictability and validity mistaking mathematical consistency for empirical truth.

Assumption of Staff Passivity

The hybrid architecture creates a structural imbalance in human agency. The ABM layer grants the student agents high cognitive complexity - they are reactive, emotional, and socially aware. Conversely, the DES layer treats the staff as passive operational components who absorb `staffFatigue` and slow down linearly. In reality, human staff possess operational agency. Under acute pressure, human operators may frequently increase their processing speed by temporarily shedding non-essential tasks, or they actively deploy interpersonal skills to de-escalate a student's frustration. By treating the service desk purely as a degrading resource pool, the model accurately captures systemic exhaustion but ignores the adaptive human resilience on the other side of the desk.

Assumption of Pooled Resource Exhaustion

Fatigue is calculated for the entire campus service node as a single, pooled variable rather than tracking individual staff member exhaustion. This means the model cannot capture granular shift dynamics where specific staff members reach cognitive fatigue limits and require breaks or rotation while others remain fresh. Tracking individual fatigue would require substantially more complex rostering logic within the DES layer, but it would produce significantly more accurate throughput modelling under sustained, high-hostility load. Basically generalising the fatigue for whole set of resources is wrong.

Linearity of Friction Transfer

The Friction Transfer mechanism is the critical bridge between the cognitive and operational layers, but its mathematical execution is strictly linear. The model transfers frustration to staff fatigue at a capped, steady rate (`rageIndex * 0.1`). Empirical studies in cognitive ergonomics suggest that operational burnout or workload is rarely a steady drip; it behaves as a step-function or a cliff. A staff member might handle twenty highly frustrated students with zero measurable degradation in processing speed, only for the twenty-first student to trigger a complete cognitive collapse. The linear accumulation model successfully simulates general network drag, but it likely underestimates the suddenness with which real-world service nodes fail.

Environmental brittleness without resilience design

The σ=12.68 at 80% adoption quantifies a real vulnerability that the current system cannot address. Weather events - or any other stochastic environmental shock that simultaneously compresses demand and routing options - can overwhelm the information layer even at high adoption. The model identifies this vulnerability precisely but does not design for it.

{#12} Future Iterations and Design Implications

Throughput Signal vs Queue Length Signal

The current shadow queue mechanism surfaces predicted load (how many agents are heading to a node) but not predicted throughput (how fast that node is currently processing). Queue length alone is an ambiguous signal: a queue of twenty agents being served at high throughput may resolve faster than a queue of eight agents at a fatigue-degraded node. A core implication for the app's future design is incorporating a dynamic throughput signal. This would make the cognitive routing engine informationally honest, allowing users to distinguish between a long queue produced by a high arrival rate and one produced by service degradation.

Dynamic Trust as an Adoption Lever

The threshold finding - that the system crosses from unstable to stable somewhere between 50% and 80% adoption - was produced under static conditions where `infoTrustScore` remained fixed. In real-world deployment, trust and adoption are dynamic variables shaped by perceived utility. A major implication is that if weather shocks produce poor recommendations during early deployment, trust will degrade, and the effective adoption rate may fall below the critical threshold. Future models must treat adoption as a dynamic, feedback-driven output of system performance rather than a fixed experimental input.

Psychologically Distinct Agent Populations

The model implicitly produces two behaviorally distinct psychological profiles, which has profound implications for how waiting is understood. App users carry a pre-arrival emotional load shaped by anticipated queue pressure; their frustration begins before they have left their origin point. Conversely, blind users arrive emotionally neutral. This maps onto the documented cognitive distinction between anticipated waiting and experienced waiting. Future iterations must make this explicit by introducing an on-arrival visual queue perception trigger for blind users at `Door_In` to capture the immediate frustration response of walking into a physically visible long queue before a single minute of waiting has elapsed.

Resilience Layer Design

The initial optimisation solved the efficiency problem - cost-minimal staffing under normal operating conditions. However, the identified brittleness highlights that efficiency and resilience are entirely different design problems. The primary implication for future architecture is the necessity of a dedicated resilience layer utilising different design instruments: emergency capacity triggers activated by the weatherMultiplier breaching a defined threshold, dynamic routing modifications that increase the balancingBias penalty under severe weather to counteract the proximity reversion effect, and real-time staff reallocation protocols.

{#13} Macro System Synthesis & Scalability

Diminishing Returns and Stochastic Volatility

Going into this project, I expected a steady, linear relationship between app adoption and breach reduction. The 1,000-replication dataset revealed a much harsher reality: the brutal curve of diminishing returns. Getting just 50% of the population to use the app crushed the baseline failure rate, dropping mean SLA breaches from 140 down to 24 (an 83% reduction).

However, at 50% adoption, the standard deviation (26.89) was actually higher than the mean. This wasn't just generic variance; it explicitly quantified the model's stochastic volatility. It proved that while the system was highly efficient on an average day, it was structurally brittle and prone to catastrophic failure under environmental shocks (e.g., severe weather events suddenly compressing routing options). Pushing adoption to 80% barely moved the needle on absolute volume-dropping the mean to 7.3 - but it was mathematically required to cut that stochastic volatility in half and stabilise the ecosystem.

Strategic Takeaway

In mission-critical service design, fixing the "sunny day" scenario is the easy part - 50% adoption achieves that. But users don't remember average days; they remember catastrophic failures during environmental shocks. The true product challenge isn't the initial rollout; it is engineering the aggressive adoption strategies and resilience layers required to capture that final 30% of the user base, which is the only way to insulate the system against stochastic volatility.

Diminishing Returns and Stochastic Volatility

Going into this project, I expected a steady, linear relationship between app adoption and breach reduction. The 1,000-replication dataset revealed a much harsher reality: the brutal curve of diminishing returns. Getting just 50% of the population to use the app crushed the baseline failure rate, dropping mean SLA breaches from 140 down to 24 (an 83% reduction).

However, at 50% adoption, the standard deviation (26.89) was actually higher than the mean. This wasn't just generic variance; it explicitly quantified the model's stochastic volatility. It proved that while the system was highly efficient on an average day, it was structurally brittle and prone to catastrophic failure under environmental shocks (e.g., severe weather events suddenly compressing routing options). Pushing adoption to 80% barely moved the needle on absolute volume-dropping the mean to 7.3 - but it was mathematically required to cut that stochastic volatility in half and stabilise the ecosystem.

Product Takeaway

In mission-critical service design, fixing the "sunny day" scenario is the easy part - 50% adoption achieves that. But users don't remember average days; they remember catastrophic failures during environmental shocks. The true product challenge isn't the initial rollout; it is engineering the aggressive adoption strategies and resilience layers required to capture that final 30% of the user base, which is the only way to insulate the system against stochastic volatility.

Individual Rationality vs Collective Failure

This project fundamentally changed how I view user behavior versus system outcomes. Every single "blind" agent in this simulation makes a perfectly rational, logic-based decision: route to the nearest campus to save travel time. No individual is behaving incorrectly. Yet, systemic failure emerges precisely from the aggregate of these individually rational decisions made under a shared information constraint.

Strategic Takeaway

You cannot fix a structurally broken ecosystem by trying to change individual human behavior. You can only fix it by changing the information architecture of the environment itself. It is an infrastructure problem, not a behavioral one.

The 50/50 spawn Failure : Designing the right stress test

The biggest breakthrough in my thinking didn't come from a success - it came from a massive failure in my early experimental setup. In my first prototype, I had agents spawning evenly with a 50/50 geographic distribution. Under this baseline, running the simulation at 0% adoption and 90% adoption yielded the exact same result: near-zero breaches. I initially assumed my routing code was broken. But after a deep audit, I realised the logic was perfect; the problem was the environment. A symmetric 50/50 spawn accidentally balanced the campus loads perfectly through simple proximity. The structural bottleneck I built the app to fix didn't actually exist under those conditions.

Strategic Takeaway

Shifting to a skewed 60/40 demand distribution wasn't an arbitrary tweak to force an interesting result; it was the only way to accurately model the real-world conditions under which the problem actually occurs. It transformed the project from a hollow validation exercise into a genuine stress test of whether information architecture can override physical demand imbalances.

Cross-Domain Generalisability

This framework proves that total system capacity is often secondary to the architecture of demand routing. The blueprint built here is directly generalisable to any high-stakes, resource-constrained service environment where demand reverts to physical proximity in the absence of real-time signals. Hospital emergency room triage, airport security lane optimisation, and logistics network routing all share the exact operational geometry engineered in this project.
