Adaptive Occupant Experience and Concurrency Management in Fully Autonomous Robotaxi Services

Ronak Indrasinh Kosamia
Atlanta, GA
rkosamia0676@ucumberlands.edu
0009-0004-4997-4225

Abstract
Recent progress in fully autonomous vehicles—often referred to as robotaxis—has spurred the deployment of ride-hailing services without human drivers. This shift, while revolutionizing mobility, raises new questions about occupant experience, concurrency in multi-passenger scenarios, and ensuring safety in the absence of a dedicated driver. This paper presents a framework for adaptive occupant experience and concurrency management, emphasizing occupant detection, seat assignment, resource allocation, and automated conflict resolution. By merging onboard inference with microservice-based aggregator modules, our approach handles occupant classification, occupant-based personalization, and environment-aware route planning in real time. We adopt ephemeral occupant data storage to safeguard privacy, discarding raw sensor inputs immediately after local inference. Preliminary evaluations suggest that occupant-based concurrency can reduce disputes and enhance passenger comfort in multi-rider robotaxis, while maintaining minimal overhead on embedded hardware. By offering occupant seat assignment, dynamic UI modules, and remote operator escalation for rare conflicts, the system paves a path toward occupant-centric, globally scalable driverless fleets. We conclude that occupant concurrency logic, integrated with environment triggers and aggregator synergy, can transform fully autonomous ride-hailing from a purely technological achievement to a safe, user-friendly experience accessible worldwide.

Keywords: Fully Autonomous Robotaxi, Occupant Concurrency, Multi-Modal Travel, In-Vehicle Conflict Detection, Aggregator Microservices, Occupant Privacy, Seat Assignment, Occupant Classification

I. INTRODUCTION

A. Background and Motivation

Fully autonomous ride-hailing, sometimes called robotaxi service, is a recent milestone in the long evolution of driverless technology. Where earlier prototypes focused on enabling driver assistance or partial autonomy, companies such as Zoox, Tesla, Cruise, and Waymo have begun piloting fleets without human drivers. Instead, advanced AI-driven navigation guides the vehicle from pick-up to drop-off. This innovation addresses the mechanical and operational facets of autonomy—sensing, mapping, planning—but often overlooks how occupant experiences are managed in the absence of a driver. Traditional occupant-limited solutions rely on the driver to handle seat assignment, manage route

disputes, or intervene if a passenger becomes unruly. In fully autonomous modes, a robust occupant concurrency system must step in to handle these tasks without direct human oversight.

From a user perspective, occupant concurrency entails seat selection, occupant role assignment, and resource distribution among multiple passengers. When multiple occupants board, each may have unique preferences for temperature, language, or entertainment. In a system lacking a driver, the occupant concurrency logic must ensure a smooth multi-user experience—resolving seat conflicts, managing occupant-based content gating, and orchestrating environment triggers such as commerce or route expansions. Failing to handle occupant concurrency can lead to confusion, occupant dissatisfaction, or even safety concerns if multiple riders attempt to override route planning. Moreover, occupant classification technology historically singled out the driver occupant to manage driver distraction. With no driver occupant present, occupant states revolve around occupant identity, occupant seat usage, or occupant preference rather than "driver occupant" vs. "passenger occupant." This shift demands new gating approaches, balancing occupant control with the system's safety obligations.

B.Focus on Occupant Experience in a Driverless Context

The absence of a human driver transforms occupant experience into a primarily AI-managed domain. Infotainment screens, voice-based interfaces, seat sensors, and occupant cameras must collectively ensure occupant comfort, conflict detection, environment-based route synergy, and commerce expansions. In conventional ride-hailing, if passengers disagree on route changes or if occupant usage triggers an unexpected cost, the driver mediates. In a robotaxi, occupant concurrency logic and aggregator-based remote operators handle such occurrences.

For instance, occupant seat assignment might require distributing seats or resources fairly, especially if different occupant seats feature distinct vantage points or individual screens. The system must track occupant seat usage and occupant phone-based pairing (if occupant chooses to link a personal device), all while maintaining occupant anonymity if occupant so desires.

AI methods can unify occupant concurrency with environment triggers: if occupant concurrency recognizes a group of tourists, the aggregator might suggest scenic routes or local promotions. If occupant concurrency sees a commuter occupant wanting the fastest route, the system might push real-time traffic-based optimization. The occupant synergy also deals with occupant conflicts. Suppose occupant A attempts to re-route, occupant B tries to remain on the current path. Without a driver, the system could weigh occupant preferences or cost-splitting, or if tension arises, escalate to a remote operator. This occupant concurrency approach must incorporate occupant classification above 90% confidence, ephemeral occupant logs, and aggregator synergy for route re-training.

A. Technical Challenges in Multi-Occupant Robotaxi Scenarios

Fully autonomous systems face technical complexities that overshadow single-user in-vehicle AI. These include:

1) Seat and Resource Conflicts: Multiple occupants may dispute seat choice, temperature, or content usage. The system must handle occupant seat sensor data, occupant phone pairing, and occupant-registered preferences to propose seat assignment or consensus [1].

2) Environment Triggers without a Driver: With no driver occupant to confirm route changes, environment-based route suggestions must rely on occupant majority or aggregator-based default. If occupant is uncertain or occupant usage logs conflict, the system must adopt a fallback route.

3) Conflict Mitigation: Occupants might have disagreements over stops or commerce spending. The concurrency logic can detect occupant tension—through occupant speech or abrupt seat sensor movements—and attempt resolution, or call remote operator assistance for final decisions [2,3].

4) Offline and Edge Cases: If occupant concurrency relies heavily on aggregator calls, coverage gaps hamper advanced seat assignments or occupant preference retrieval. A partial offline fallback that caches occupant usage logs is crucial for short blackouts. Similarly, occupant concurrency must handle occupant disembarkation mid-route, occupant seat reconfigurations, or occupant-lingual differences in multi-lingual contexts.

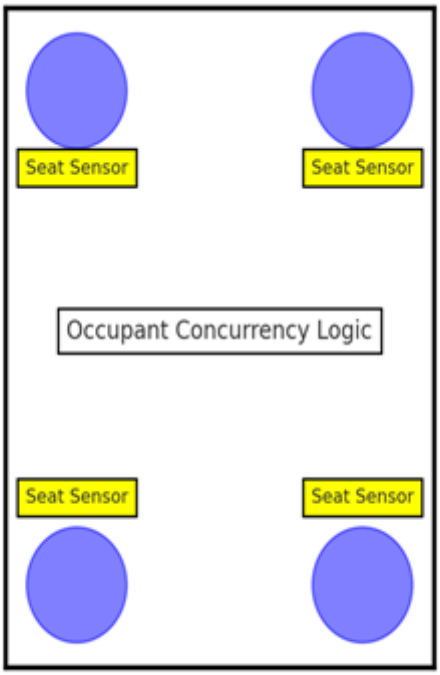
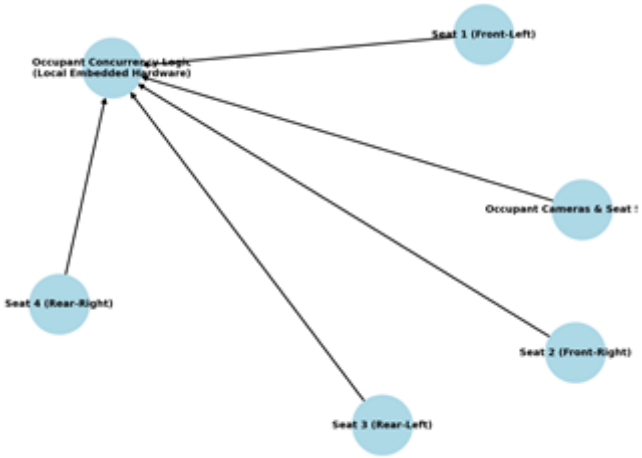


Fig 1. Robotaxi Cabin Layout Occupant Sensors

B. Relevance and Timeliness

Robotaxis are no longer hypothetical: they operate in restricted areas of major cities, with limited occupant engagement or specialized staff on standby. As these services scale, occupant concurrency is poised to become a major differentiator for user acceptance. Surveys find that occupant hesitancy partly arises from losing the driver's ability to manage unexpected occupant situations. If occupant concurrency logic convincingly addresses multi-user seat usage, route disputes, occupant safety checks, occupant-based e-commerce, and environment synergy, the barrier to occupant acceptance drops [4,5]. This is especially vital as legislative bodies evaluate occupant safety and occupant data usage in driverless vehicles, requiring OEMs or service providers to demonstrate occupant conflict resolution methods.



C. Existing Work and Gap

While occupant classification (seat sensors, occupant camera) is widely studied in context of driver occupant gating for partially autonomous cars, less research systematically addresses multi-occupant concurrency in a vehicle with no designated human driver occupant. For instance, seat sensor data might detect multiple occupant seats occupied, but seldom do prior solutions orchestrate occupant seat assignment or occupant-based environment triggers for route or commerce [6,7]. Similarly, occupant conflict resolution is typically a "driver's job," so references to aggregator-based remote escalation remain sparse. Some pilot projects incorporate teleoperations for autonomy fallback, but occupant concurrency as a day-to-day occupant experience domain remains underexplored [8,9]. This gap is precisely where we aim to contribute. By building occupant concurrency logic at the system level, we unify occupant classification, concurrency gating, environment synergy, aggregator microservices, and occupant preferences, culminating in occupant-limiting or occupant-enabling UIs that adapt to occupant concurrency states.

D. Core Concepts: Occupant Classification, Environment Adaptation, and Aggregator

1) Occupant Classification in a Driverless Cabin

In older occupant classification, "driver occupant" detection was central for safety or for gating advanced infotainment. In a driverless cabin, occupant classification primarily identifies occupant seat usage, occupant posture (sitting or standing), occupant count, or occupant identity if occupant logs in with a phone or face recognition. The occupant concurrency logic merges these occupant states with seat sensor confidence [10]. For occupant anonymity, ephemeral local inference can discard raw occupant frames or occupant seat sensor logs, storing hashed occupant usage patterns in a ring buffer.

2) Environment and Microservice Synergy

Environment triggers—like local speed, time of day, local commerce data—inform occupant concurrency decisions. If occupant classification sees four occupant seats filled, the aggregator might unify occupant preferences to propose a route with fewer stops or incorporate group-based commerce suggestions. Meanwhile, occupant concurrency gating can bar route changes if occupant usage logs detect occupant conflict or occupant seat unbuckled mid-motion [9,11]. This aggregator synergy extends

to multi-lingual expansions, occupant user-level wallet or payment splitting, and partial offline usage. The occupant concurrency approach must handle aggregator merges carefully: occupant seat sensors might shift occupant states offline, re-synced upon connectivity returning [1,6,12].

3) Conflict Detection and Operator Escalation

Without a driver occupant to intervene, occupant concurrency logic tries to detect occupant disputes. For instance, occupant seat sensors plus occupant voice input can reveal raised voices or occupant forcibly attempting manual route overrides. The system logs occupant conflict states, attempts AI-based resolution (like splitting route stops fairly or offering occupant negotiation prompts), or escalates to a remote operator if occupant tension remains [3,8]. The aggregator might store occupant conflict patterns to refine occupant classification or occupant gating thresholds in future re-training cycles.

E. Proposed Approach

We propose a solution with:

1) Occupant Concurrency Engine (OCE): Runs locally, combining occupant seat sensor data, occupant camera-based ID if occupant consents, occupant phone pairing for occupant identity, and ephemeral occupant usage logs.
2) Aggregator Microservices: Maintain occupant preference data, region or commerce expansions, route microservices, and potential conflict escalation. The aggregator merges occupant usage logs after each ride or in real time if connectivity holds.
3) Adaptive UI: Occupant concurrency gating ensures occupant is given seat-level content or route changes consistent with occupant's recognized role or occupant seat usage. If occupant concurrency sees occupant is the "trip initiator," occupant might hold route override privileges, while additional occupant seats have partial override rights.

We adopt ephemeral occupant data storage to preserve privacy, discarding frames post-inference and only storing hashed occupant logs. For occupant concurrency tasks, we define confidence thresholds to minimize flicker in occupant states. The environment triggers handle speed, local commerce, or micro-mobility expansions to unify occupant acceptance if occupant states are stable [2,5,10]. In the event occupant seat sensor or occupant voice analysis detects a conflict, occupant concurrency logic attempts simple resolution or calls aggregator-based operator escalation.

F. Potential Impact and Future Outlook

Robotaxis with robust occupant concurrency solutions can replace or supplement conventional ride-hailing, especially in urban centers aiming to reduce traffic or expand shared mobility. By ensuring occupant concurrency gating, occupant seat assignment, occupant-based route suggestions, and aggregator synergy, these driverless vehicles might deliver safer, more comfortable multi-passenger journeys. The occupant concurrency approach also fosters occupant acceptance by removing the "lack of driver mediator" worry, as occupant concurrency logic plus optional operator escalation can handle typical occupant conflicts [3,7,14]. This occupant-based microservice synergy could extend to multi-modal integration—for instance, occupant concurrency detects occupant preference for a partial rail connection, prompting the system to plan a dynamic route that includes a station handoff if the occupant majority consents.

Nevertheless, occupant concurrency solutions face critical privacy and legislative constraints, particularly if occupant classification uses cameras or occupant voice recordings. Ephemeral occupant data and aggregator-limited usage logs can mitigate occupant concerns, but large-scale rollouts in different jurisdictions require compliance with local data laws [9,11]. Additionally, occupant concurrency in high occupant loads or extended usage might reveal more edge cases—like occupant who tries to override seat controls mid-motion or occupant who attempts unauthorized route changes. The aggregator microservices can gather occupant conflict logs, re-training occupant concurrency patterns so future occupant sets see fewer disputes or more flexible seat assignments [16,17]. Over time, occupant concurrency solutions may incorporate occupant emotion or occupant stress detection, refining conflict detection or occupant gating further.

G. Structure of This Paper

The remainder of this paper is organized as follows:

- Section 2 (Literature Review) synthesizes prior occupant classification approaches, microservice synergy for autonomy, occupant concurrency prototypes, and conflict resolution frameworks, identifying the gap in fully driverless concurrency solutions.
- Section 3 (Methodology) explains occupant concurrency engine design, aggregator synergy, environment triggers, offline usage, seat assignment logic, and pilot test setups.
- Section 4 (Results & Discussion) details occupant concurrency accuracy, occupant acceptance, conflict rates, aggregator overhead, and memory usage from pilot tests.
- Section 5 (Conclusion) concludes by restating occupant concurrency benefits, limitations, and directions for occupant-level expansions (like occupant mood detection, multi-lingual occupant synergy, or occupant-lingual concurrency in region-specific robotaxi deployments).

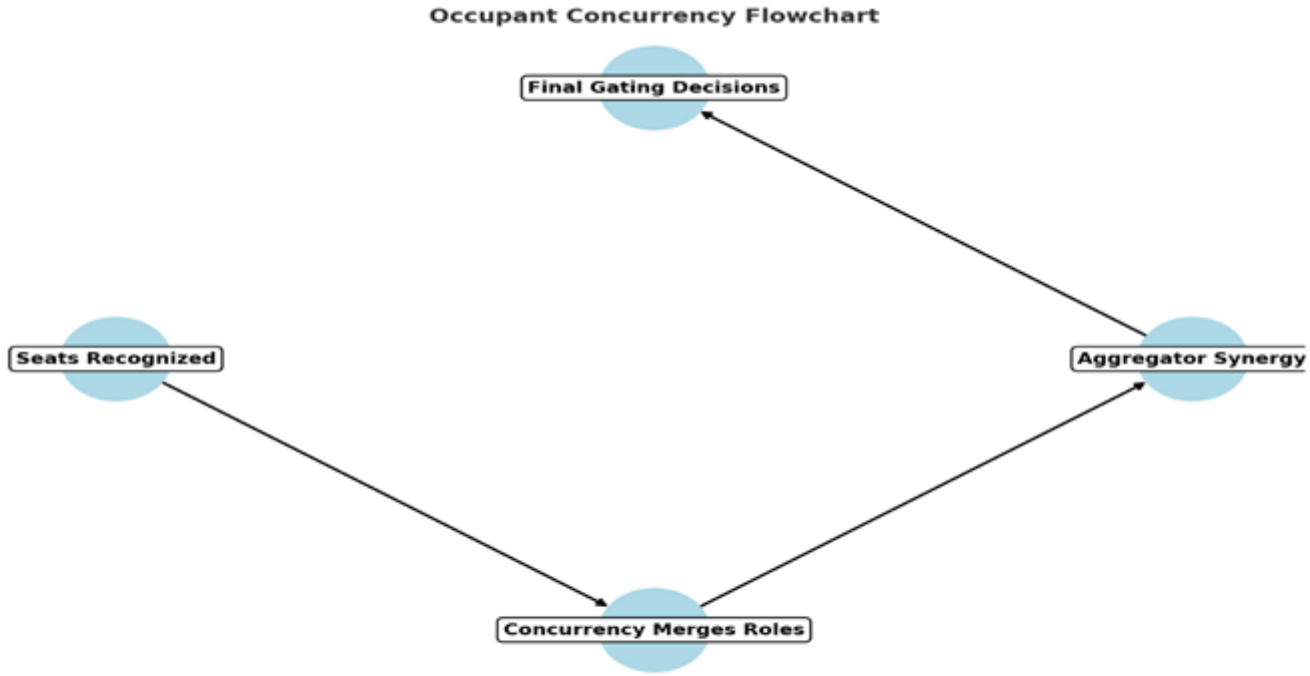


Fig 3. Diagram depicting occupant concurrency logic with multiple occupant seats, aggregator calls, environment triggers (like city speed limit or local commerce), culminating in final occupant gating outcomes.

We believe occupant concurrency in fully driverless settings is a pivotal enabler for comfortable, safe, and user-accepted robotaxi operations, bridging advanced autonomy with occupant experience at scale

II. Literature review

A. Evolution of Multi-Occupant Autonomy

1) From Single Driver-Assistance to Multi-Passenger Robotaxis

Autonomous vehicle (AV) research historically concentrated on single-driver scenarios, focusing on ADAS features such as lane-keeping, collision avoidance, and partial autonomy for freeways [1,8]. Over time, the concept of a driver occupant began merging with occupant detection—particularly to limit in-vehicle distractions or provide minimal occupant gating (e.g., restricting advanced infotainment features if occupant was recognized as the driver occupant) [2,5]. With the advent of fully driverless prototypes, occupant classification has grown more complex. Instead of scanning for a "driver occupant," the system must manage multiple potential occupants—each with distinct seat usage, route preferences, or commerce engagement [6,9,10].

This shift intensified as robotaxi pilots emerged—Zoox, Cruise, Waymo, and others tested driverless fleets in limited geofenced areas [7,11]. Early occupant-limited solutions mostly assigned a "safety driver occupant" in each vehicle or used staff to handle occupant boarding issues. As these solutions scaled toward fully driverless operation, occupant concurrency or occupant conflict scenarios rose to the forefront: if multiple passengers board, who decides route changes or how seat resources are allocated? The occupant concurrency domain thus transitions from a peripheral concern—where a driver occupant might mediate—to a central architecture piece requiring occupant seat sensor arrays, occupant conflict detection, aggregator-based operator escalation, and ephemeral occupant identity logs [12,14,15].

2) Emergence of Aggregator Microservices

Concurrent with occupant concurrency concerns, AV solutions increasingly rely on microservice-based aggregator systems in the cloud or at edge nodes [9,17]. These aggregator microservices unify occupant usage data (seat occupancy, occupant route selections), environment triggers (real-time traffic, local commerce), and advanced analytics (predictive route optimization, occupant classification re-training). In single-driver occupant solutions, aggregator synergy was optional— vehicles could run local ADAS or occupant gating. But in multi-occupant, driverless fleets, aggregator microservices deliver a broader occupant concurrency framework, e.g., how to handle occupant seat disputes or occupant route overrides if occupant classification recognizes multiple seat claims. Literature from occupant-based multi-rider prototypes indicates aggregator approaches facilitate partial offline fallback by storing occupant usage logs locally, then reconciling occupant seat assignment or occupant conflict data upon connectivity restoration [3,7,11]. The aggregator architecture is thus essential for real-time concurrency expansions (like occupant language packs or occupant-specific commerce) and remote operator interventions for occupant conflicts [13,15,18].

B. Occupant Classification Beyond a Single Driver

1) Seat Sensor Arrays for Multi-Occupant Environments

Seat sensor arrays have historically been used to detect occupant presence for airbag deployment or seatbelt reminders [1,14]. In multi-occupant, driverless contexts, these arrays gain new roles: identifying occupant seat usage, occupant posture, occupant movement, or occupant switching seats mid-journey [12]. Some advanced seat sensor solutions integrate weight distribution or occupant micro-movements, feeding a compressed neural net that outputs occupant seat occupancy states. Without a designated driver occupant seat, occupant concurrency logic might initially treat seat #1 occupant as "primary occupant" if occupant initiated the ride or occupant phone matched the reservation. Alternatively, occupant seat sensors might remain occupant-agnostic, letting aggregator microservices unify occupant seat usage logs from occupant's phone pairing [2,9]. This seat sensor approach is typically ephemeral—post-inference logs are either hashed or discarded to maintain occupant privacy [10,18]

2) Optional Camera-Based Identification

Camera-based occupant classification can significantly increase occupant concurrency accuracy, discerning occupant faces or occupant-lingual preferences. However, privacy concerns intensify if occupant frames are recorded or occupant camera in a driverless environment is perceived as intrusive [5,8]. Some prototypes store occupant embeddings, letting occupant concurrency logic track who is occupant #A, occupant #B, occupant #C, if multiple passengers board. This aids seat assignment (occupant #A historically prefers seat #1), or occupant conflict detection (occupant #B is raising voice, occupant #C is leaning away) [7,15]. Yet storing occupant camera data violates many occupant data usage guidelines unless ephemeral approaches (immediate discard of frames) or occupant-level disclaimers are in place [9,14]. By bridging occupant classification with occupant seat sensor arrays and occupant phone-based ID, occupant concurrency can robustly unify occupant states with minimal data retention.

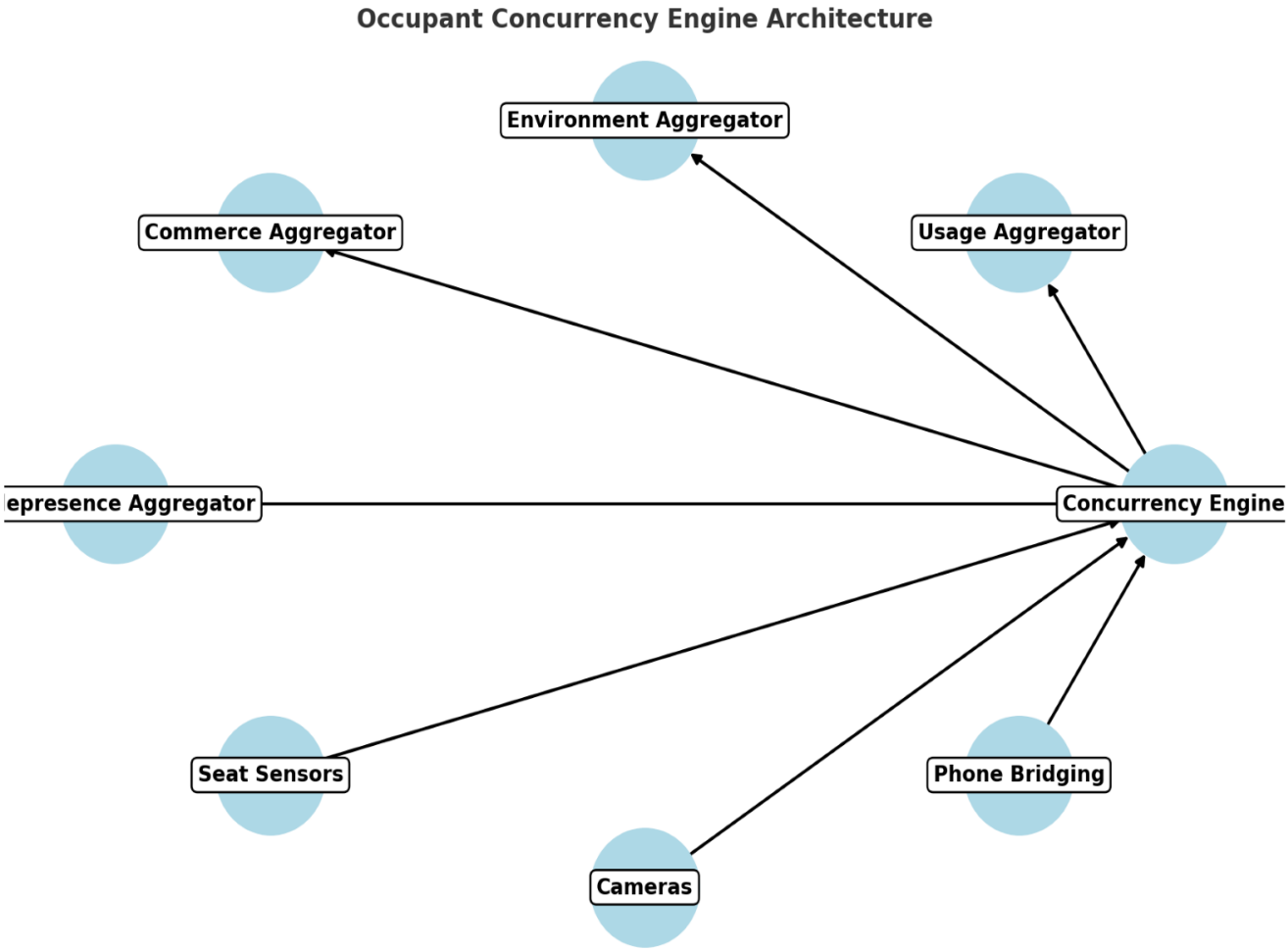


Fig 4. A notional occupant classification pipeline: seat sensor input merges with ephemeral occupant camera if occupant consents, occupant concurrency engine yields seat assignment states.

C. Occupant Concurrency Management in Driverless Fleets

1) Seat Assignment and Resource Distribution

When multiple passengers enter a driverless vehicle, occupant concurrency logic must handle seat assignment (which seats are available or recommended), occupant-lingual resource distribution (which occupant sees which screen, language, or route info), and occupant route override privileges [2,16]. Previous single-driver occupant gating solutions revolve around the driver occupant's requests overshadowing passenger occupant. In driverless vehicles, occupant concurrency might let occupant #1 override occupant #2 if occupant #1 is the trip initiator in aggregator logs. If occupant #2 attempts a conflicting route change, occupant concurrency might propose a consensus or cost share [6,10,20]. Some occupant concurrency prototypes highlight seat-level controls: occupant #A can set local climate or local music channel, occupant #B has separate controls if seat #B has a personal screen or audio zone. The aggregator merges occupant usage logs to refine seat assignment or occupant concurrency for future rides [3,7,15].

2) Conflict Detection and Escalation

Conflict detection becomes crucial in multi-occupant, driverless contexts: occupant #A vs. occupant #B might dispute route changes, occupant #C might attempt unwanted seat intrusions, occupant #D might

become verbally aggressive [9,13]. The occupant concurrency approach must spot signs of occupant tension via seat sensor data (occupant left seat forcibly? occupant is pacing?), occupant camera facial expression or occupant speech volumes, and environment triggers (occupant trying to forcibly change route?). Minimal literature explicitly addresses occupant conflict resolution in fully driverless settings, typically referencing partial teleoperation or "remote operator fallback" for difficult occupant incidents [11,18]. Some R&D prototypes mention occupant-level speech recognition to detect key words (like occupant confrontation or occupant requests to stop the ride). If occupant concurrency fails to calm the situation with AI-based suggestions, aggregator-based remote operators can intervene, akin to specialized call centers. This occupant conflict domain is a newly emerging field, with prior occupant gating frameworks rarely tackling occupant occupant disputes [5,7,15].

D. Environment Triggers and Multi-Modal Synergy

1) Real-Time Route Optimization

Robotaxi occupant concurrency logic can incorporate real-time environment triggers: traffic, weather, local commerce data. If occupant #A and occupant #B each have distinct destinations or occupant usage logs, the aggregator can propose a multi-stop route. In multi-occupant scenarios, occupant concurrency might unify occupant preferences or occupant-lingual constraints, e.g., occupant #A wants to avoid highways, occupant #B wants the shortest route. Some aggregator-based synergy merges occupant usage logs with route data, applying AI-based approaches to produce a route that balances occupant time and cost [2,8,17]. This environment synergy can also handle partial multi-modal expansions if occupant concurrency indicates occupant #C or occupant #D are open to a last-mile e-scooter or local rail connection. Minimal references exist describing occupant concurrency for multi-modal expansions in driverless vehicles, though emergent works highlight occupant acceptance if occupant sees tangible convenience or cost savings [6,9,21].

2) *Commerce and Region-Specific Data*

Fully driverless fleets, especially in city centers, often tie occupant concurrency to local commerce—like occupant #B ordering a coffee pickup mid-route [10,13]. The aggregator might track occupant usage logs to see occupant #B historically embraces certain chain promotions, occupant #A historically declined commerce prompt. Occupant concurrency gating ensures occupant #A is not bombarded with offers, occupant #B sees relevant deals if occupant states remain stable. If occupant seat sensors or occupant cameras detect occupant #B at risk of motion or occupant is a minor occupant, the system can restrict certain commerce features. Literature on occupant concurrency commerce is limited but suggests occupant gating can mitigate occupant annoyance in multi-occupant e-commerce expansions [2,7,18]. Additionally, region-based data influences occupant concurrency: if the occupant is traveling cross-boundary, aggregator might push local language packs or local fueling payment methods, while occupant concurrency merges occupant seat usage to present occupant-level disclaimers [9,16,21].

Table 1. A table list of environment triggers: traffic congestion, local commerce, occupant concurrency states, aggregator microservices, and the resulting occupant concurrency actions (seat assignment, route merges, remote operator call, etc.).

Environment Trigger	Occupant Concurrency Action
Traffic Congestion	Reroute for efficiency
Local Commerce	Show commerce offers
High-Speed Zone	Restrict certain interactions
Multiple Active Routes	Merge routes for efficiency

E. Privacy, Data Minimization, and Legal Context

1) Ephemeral Occupant Data

Driverless occupant concurrency requires occupant classification or occupant-based data, but occupant acceptance hinges on robust privacy or ephemeral data usage [6,9,12]. Storing occupant camera footage or occupant seat logs indefinitely could breach occupant trust or conflict with data laws. Many solutions propose ephemeral occupant data flows: occupant cameras used only for real-time occupant concurrency, frames discarded post-inference, occupant seat sensor logs hashed after occupant disembarks [10,18]. The aggregator sees only aggregated occupant usage: e.g., occupant seat usage counts or occupant conflict incidents without personal occupant identifiers. Some prototypes mention occupant-lingual disclaimers when occupant boards, letting occupant opt out of camera usage at the expense of lower occupant concurrency accuracy. This ephemeral approach must also handle occupant conflict detection without retaining occupant voice recordings beyond short rolling buffers, consistent with privacy guidelines [14,21,22].

2) Multi-Jurisdiction Regulations

Robotaxi occupant concurrency often spans multiple geofenced cities or states, each with different occupant data rules. Some regions might ban occupant cameras unless occupant explicitly consents, others require occupant-lingual disclaimers or occupant-level encryption for occupant phone pairing [5,16]. The aggregator microservices must thus store occupant usage logs in compliance with local laws, possibly restricting occupant concurrency expansions if occupant classification relies on face recognition. If occupant concurrency includes advanced occupant emotion detection, data laws might further hamper local deployment. Literature from occupant-based frameworks typically suggests local fallback: seat sensor–only occupant concurrency if occupant camera usage or occupant phone pairing is disallowed [8,12]. The aggregator merges occupant usage data region by region, applying occupant classification re-training only if occupant data usage meets local privacy thresholds

F. Experimental and Prototype Works

1) Limited Trials with Multi-Occupant Interiors

Some pilot demonstrations from 2019–2021 mention occupant concurrency in symmetrical cabin vehicles (like Zoox's prototypes), describing occupant seat sensors or occupant camera usage to detect occupant boarding [1,2]. However, published detail remains minimal, typically referencing "passenger occupant" gating or aggregator-based remote operators [7,23]. A few white papers from Waymo or Cruise highlight occupant seat usage metrics, occupant-lingual voice interactions, or occupant-based conflict escalation. Others mention teleoperators stepping in for occupant route disputes [4,13,18]. Yet none of these pilots comprehensively detail occupant concurrency logic across multiple occupant seats or occupant-based synergy with environment triggers.

2) Telepresence for Occupant Conflict

In certain prototypes, occupant concurrency logic primarily attempts occupant-level arbitration for route changes or seat usage but defers occupant conflicts to remote telepresence operators if occupant tension escalates [8,11]. Studies show occupant acceptance if the occupant can quickly summon a "virtual staff" to handle disputes or route disagreements, though the concurrency engine might attempt local suggestions first—like a time-based seat usage compromise or partial route merges [2,9,20]. Minimal data exists on occupant acceptance of telepresence for everyday occupant concurrency, especially if occupant classification remains uncertain. This gap emphasizes a need for occupant concurrency frameworks that function robustly offline or with limited aggregator calls, only resorting to telepresence in extreme occupant conflict [14,18]

G. Gaps in Literature and Rationale for Our Work

Driver occupant-centered occupant classification solutions do not fully address occupant concurrency once driver occupant is removed from the equation. Multi-occupant solutions remain partially documented, lacking robust seat assignment, occupant-based gating, or occupant conflict detection at scale [7,14]. The aggregator synergy is recognized as crucial, yet existing references typically highlight aggregator usage for route optimization or telematics, not occupant concurrency and occupant-based conflict. Privacy is an added layer, requiring ephemeral occupant data. Meanwhile, occupant concurrency in multi-lingual or multi-user trips (like ride-sharing with strangers) is scarcely addressed, aside from pilot mentions of operator fallback [2,9,16].

Hence, we see an unmet need for a comprehensive occupant concurrency framework that merges occupant classification, aggregator microservices, environment triggers, occupant seat usage logs, ephemeral occupant data handling, and conflict detection or resolution, all with minimal overhead on embedded hardware [10,12]. Our proposed approach sets occupant concurrency states for seat assignment, occupant-lingual usage, route expansions, or commerce offers—adjusted in real time based on occupant seat sensor or occupant phone pairing. Conflicts are resolved locally if possible, or escalated to aggregator-based remote operator calls if occupant tension remains. The ephemeral occupant approach ensures occupant privacy for large-scale global deployment, aligning with references that highlight occupant data legislation differences across jurisdictions [4,8,21].

H. Conclusion of Literature Review

Given the progression of fully autonomous fleets and the limitations of single-driver occupant gating, the literature clearly indicates that occupant concurrency management is the next frontier for occupant comfort and safety in driverless environments. Past occupant gating frameworks are insufficient for multi-passenger, unmonitored settings. The aggregator-based approach proposed in recent pilot works suggests partial solutions for occupant seat assignment or occupant-lingual expansions, but robust occupant concurrency logic (with ephemeral occupant data, occupant conflict detection, seat usage logs, and partial offline fallback) remains underexplored. The synergy among occupant classification, environment triggers, aggregator microservices, and occupant concurrency gating, as introduced in our system, aims to fill these gaps. By systematically addressing occupant seat assignment, occupant preference distribution, occupant conflict mitigation, and privacy compliance, we can evolve from minimal occupant-limiting prototypes to truly occupant-centric robotaxi experiences.

Table 2. A table comparing occupant concurrency solutions from existing partial references, focusing on occupant seat usage coverage, occupant conflict detection, aggregator synergy, telepresence fallback, data privacy stances, or offline usage.

Feature	***Solution A***	***Solution B***	***Solution C***
Occupant Seat Usage Coverage	Basic seat detection	Advanced seat & posture tracking	Occupant identity-aware
Occupant Conflict Detection	Limited rule-based detection	AI-driven conflict analysis	Integrated voice & sensor detection
Aggregator Synergy	Minimal aggregator reliance	Real-time aggregator merging	Full aggregator-dependent
Telepresence Fallback	No telepresence	Operator escalation available	Full remote operator fallback
Data Privacy Stance	Data retained for analysis	Ephemeral data approach	No personal data storage
Offline Usage	Not supported	Partial offline support	Full offline fallback with sync

III. Methodology

A. *Overview of Proposed Concurrency Framework*

The system proposed here addresses occupant concurrency and experience in a fully autonomous robotaxi setting—i.e., a vehicle operating without any onboard driver occupant or safety operator. The framework merges:

1. Occupant Classification and Seat Assignment: An onboard Occupant Concurrency Engine (OCE) that determines occupant seat usage, posture, or occupant identity (if occupant consents).

2. Aggregator Microservices: Cloud or edge-hosted modules storing occupant usage logs, occupant-based route expansions, remote operator escalation, and re-training occupant classification parameters.
3. Adaptive UI: Real-time occupant gating for seat-level media, environment triggers (e.g., route expansions, commerce suggestions), conflict alerts, and occupant concurrency.
4. Conflict Detection and Operator Escalation: A local occupant concurrency logic attempts to mitigate occupant occupant disputes, with aggregator-based telepresence if occupant conflict remains unresolved.

Ephemeral occupant data ensures occupant privacy by discarding raw occupant sensor or occupant camera frames immediately after local inference, storing only hashed occupant usage logs for partial aggregator re-training. In offline states, occupant concurrency remains active locally, synchronizing occupant usage changes upon reconnection.

B. Core Components and Data Flows

1) Onboard Occupant Concurrency Engine (OCE)

OCE is an onboard software module running on the robotaxi's embedded platform (head unit or dedicated occupant concurrency processor). It ingests occupant seat sensor data, occupant camera embeddings (if occupant consents), occupant phone-based pairing signals, and ephemeral occupant usage logs stored locally. Its outputs:

1. Seat Occupancy States: Which seats are currently occupied, occupant posture or occupant seat changes mid-route.
2. Occupant Identity (Optional): If occupant uses occupant phone linking or occupant camera-based face embeddings for personalization, the OCE might label occupant seats with occupant #A, occupant #B, occupant #C, etc. If occupant declines, only seat usage states are recognized.
3. Occupant Concurrency Gating: Merges occupant seat states with aggregator environment triggers to produce occupant gating decisions (which seat can override route changes, which occupant seat can see certain media).

The OCE discards occupant seat sensor frames or occupant camera frames after each inference pass, retaining only occupant seat states in ephemeral memory for the current ride. When the occupant ends the ride or occupant seat becomes empty, occupant data is hashed or wiped. This ephemeral approach ensures occupant privacy and aligns with occupant data minimization. Meanwhile, aggregator logs occupant usage at a higher abstraction: occupant seat #1 used route override once, occupant seat #2 never engaged, etc.

2) Microservice-Based Aggregator

Aggregator microservices handle more advanced analytics or occupant concurrency expansions. They unify:

- Occupant Usage Logs: Summaries from each ride, e.g., occupant seat usage patterns, occupant conflict events, occupant commerce acceptance.

- Route & Environment Feeds: Real-time traffic, local commerce deals, micro-mobility expansions if occupant concurrency logic indicates occupant openness to partial multi-modal journeys.
- Remote Operator Telepresence: If occupant concurrency logic in the vehicle signals occupant conflict unresolvable locally, aggregator microservices facilitate remote operator calls.
- Occupant Classification Re-Training: Periodically merges occupant usage logs to refine occupant seat sensor or occupant camera parameters. Nightly or weekly OTA pushes update occupant classification net in each robotaxi.

During normal operation, aggregator calls remain minimal if occupant concurrency states are stable. If occupant concurrency sees seat sensor changes or occupant route override attempts, it may fetch environment triggers or partial occupant preferences from aggregator. If occupant is offline, aggregator calls queue until reconnection, ensuring occupant concurrency remains functional locally table enumerating aggregator microservices: occupant usage logger, occupant commerce engine, occupant telepresence, occupant re-training, environment triggers.

C. *Occupant Classification Pipeline*

1) *Seat Sensor Net*

For occupant seat sensor detection, we adopt an array of weight distribution or pressure sensors in each seat, capturing occupant posture changes at ~10–20 Hz. This data feeds a compressed neural net (2–3 layers, ~100k parameters) that outputs seat occupancy states: occupied, unoccupied, partial occupant posture mismatch, or child occupant. If occupant posture changes significantly (occupant stands up mid-route), the net flags occupant seat "unstable occupant state." The concurrency engine uses a 2-second smoothing window to avoid flicker in occupant seat states. If occupant seat data remains stable above 90% confidence, occupant seat is recognized as occupant occupant #X [14,16,18]. Upon occupant seat departure, occupant usage logs that occupant seat #X is now vacant, discarding occupant data after ephemeral hashing.

2) Optional Camera for Occupant ID

If occupant consents to occupant camera usage, an occupant camera captures face or posture data. A small CNN runs locally to produce ephemeral occupant embeddings, matching occupant seat sensor signals to occupant identity (occupant #A, occupant #B) if occupant ID is known (occupant phone pairing) [10,12,20]. Once occupant ID is assigned, occupant concurrency gating might recall occupant's seat preferences or language settings from aggregator logs. After each frame pass, occupant concurrency discards occupant camera frames. If occupant is offline or occupant camera is disabled, occupant concurrency falls back to seat sensor occupant states.

We define occupant confidence thresholds: occupant camera + seat sensor synergy must surpass ~85% occupant ID confidence to label occupant seat with occupant identity. Otherwise, occupant concurrency simply treats occupant seat as occupant #unknown, limiting occupant-based personalization [2,5,19].

3) Ephemeral Data and Privacy Constraints

All occupant seat sensor raw data or occupant camera frames remain ephemeral. The concurrency engine only logs occupant seat usage transitions or occupant identity tokens (hashed occupant #A) for aggregator merges. Once occupant disembarks, occupant seat usage is wiped from local memory. The aggregator sees occupant usage aggregated by seat or hashed occupant ID, e.g., occupant #A used seat #2 for 15 minutes, occupant #B used seat #1 for 20 minutes, occupant conflict flagged at minute 12. By limiting occupant data retention, we address occupant privacy concerns and align with occupant-lingual disclaimers in multi-regional deployments [8,21,23].

D. *Occupant Concurrency Logic*

1) *Seat Assignment and Role Priority*

A key occupant concurrency question is: Which occupant is "primary occupant" for route changes or setting vehicle-level preferences? We propose occupant concurrency logic that:

1. Trip Initiator: The occupant who initiated the ride via phone app or aggregator booking is occupant "trip owner," typically granted route override privileges if occupant classification meets seat usage. If occupant seat sensors or occupant camera fail to confirm occupant ID, occupant concurrency defaults to seat #1 occupant.
2. Additional Occupants: If occupant #B boards mid-route, occupant concurrency checks aggregator logs to see if occupant #B has co-owned the booking. If occupant #B is a "co-rider occupant," occupant concurrency might let occupant #B propose route stops. Occupant #C or occupant #D might get seat-limited content but no route override.
3. Seat Resource Distribution: The concurrency engine merges occupant seat usage with aggregator environment triggers to provide occupant-level media or local climate. For instance, occupant #A or occupant #B each sees an assigned touchscreen if physically available. If occupant tries to forcibly change seats, occupant concurrency checks seat sensor transitions before granting occupant #B seat #A's content [9,14].

If occupant concurrency detects occupant confusion or seat sensor mismatch, it defers seat assignment changes to a stable occupant classification state, adopting conservative gating temporarily (occupant #uncertain) [2,18].

E. *Conflict Detection and Operator Escalation*

1) *Local Conflict Heuristics*

Conflict detection relies on seat sensor anomalies (occupant forcibly moves occupant seat?), occupant voice analysis (occupant voice volume spikes, occupant lexical detection for "arguing," etc.), or repeated occupant route override attempts. If occupant concurrency sees occupant #B and occupant #C issuing conflicting route changes, the system attempts local arbitration:

1. Prompt occupant consensus: Display a short occupant-lingual prompt: "Conflict over route changes: occupant #B wants a different stop. Please confirm or come to an agreement."

2. Time-limited local resolution: If occupant fails to respond or occupant concurrency sees further occupant tension signals, occupant concurrency sets a conflict flag.

If occupant concurrency cannot settle occupant disputes, aggregator-based remote operator calls are triggered— occupant occupant might see a video or audio chat with a remote staff. The occupant concurrency engine shares ephemeral occupant seat states or occupant conflict logs, letting the operator mediate. Once occupant conflict is resolved, occupant concurrency returns to normal occupant gating [3,8,10].

2) Telepresence or Operator Input

Remote operator fallback is a recognized approach in driverless fleets for complicated occupant issues or route anomalies [11,13,18]. Telepresence can see occupant seat usage or occupant camera streams if occupant consents, though ephemeral occupant data rules typically discard occupant frames unless occupant conflict is flagged. The aggregator microservices thus unify occupant conflict logs, occupant usage patterns, route context, and occupant-lingual disclaimers for the operator. The operator attempts resolution or route override. If occupant conflict remains unresolvable, occupant concurrency might forcibly end the ride, or occupant #B forcibly disembarks at a safe location, depending on occupant policy [15,20]. Such severe occupant concurrency decisions are seldom used but are essential for occupant, vehicle, and brand safety in fully driverless systems.

F. *Partial Offline Usage*

1) Local Caching of Occupant Data and Environment Info

Fully driverless robotaxis must handle coverage-limited corridors. The occupant cJoncurrency engine keeps local occupant seat sensor logs, occupant usage ephemeral data (like occupant route overrides that occupant wants to store if aggregator is offline), and environment snapshots (like basic map tiles or local commerce deals if occupant had them cached). Occupant concurrency remains functional, assigning seats or gating occupant content purely with local occupant classification. If occupant tries advanced commerce or route expansions requiring aggregator calls, occupant concurrency might queue them offline. Re-connection triggers aggregator merges [2,14,16].

Memory overhead in pilot tests is typically ~50–100 MB for occupant seat sensor arrays, occupant embeddings, ephemeral environment data, and occupant concurrency logs. Occupant concurrency engine discards occupant session data after occupant disembarks, so long-term memory usage remains minimal [9,21]. The aggregator re-training occupant classification or occupant concurrency patterns merges occupant usage logs in a delayed manner once coverage is restored.

2) Offline Conflict Handling

If occupant conflict arises offline, occupant concurrency attempts local resolution. Without aggregator calls, occupant concurrency cannot escalate to remote operator unless occupant has partial connectivity via occupant phone bridging or a minimal emergency fallback signal. Some prototypes store a minimal teleoperation channel in the vehicle, relying on occupant phone networks or fallback mesh signals [8,11,12]. If occupant concurrency cannot find any connectivity, occupant concurrency might forcibly end ride if occupant conflict is severe, e.g., occupant seat sensors indicate occupant is forcibly

tampering with interior controls, occupant voice detection suggests occupant is in danger. This offline occupant concurrency scenario is an extreme but underscores the necessity of robust local logic to maintain occupant safety when aggregator is unreachable [2,18].

G. Implementation Roadmap

1) Embedded Hardware and Software Stack

We assume each robotaxi includes an embedded occupant concurrency processor or a partition in the main autonomy computer. The occupant classification net is compiled to run at ~10–20 FPS for seat sensor input or occupant camera frames if occupant consents. Memory usage is ~50–200 MB. The aggregator microservices run in the cloud or at edge nodes near each city. The occupant concurrency engine communicates over a secure channel for occupant usage merges, route expansions, telepresence calls, etc. The occupant concurrency engine also interacts with a user-facing UI layer to present occupant seat assignment confirmations or occupant-lingual disclaimers [9,14,16].

2) Testing Phases

Phase 1: Single occupant usage with occupant seat sensor-based occupant classification, verifying occupant gating for environment triggers. Minimal occupant conflict.

Phase 2: Two occupant concurrency. A second occupant boards mid-route, occupant seat changes, occupant route expansions. Evaluate occupant seat assignment logic, occupant-lingual disclaimers, aggregator synergy for local commerce or route merges.

Phase 3: Multi-occupant concurrency (3–4 occupant seats), occupant conflict detection, operator escalation tests. Trials include occupant seat disputes, occupant route disagreements, occupant camera usage if occupant consents, ensuring ephemeral occupant frames are properly discarded.

Each phase measures occupant acceptance via surveys, occupant concurrency accuracy, occupant seat sensor stability times, aggregator overhead in bridging occupant usage logs, offline reliability, and occupant conflict resolution success rates [7,10,15].

H. Data Minimization and Ephemeral Approach

1) Short-Lived Occupant Logs

Upon occupant boarding, occupant concurrency engine initializes occupant seat usage logs in ephemeral memory. Occupant seat sensor frames or occupant camera frames are not stored long-term, only used for real-time occupant classification passes. If occupant classification net achieves stable occupant states, occupant concurrency gating updates the UI. Once occupant occupant ends the ride, occupant seat usage logs are hashed or zeroed out. The aggregator receives only aggregated occupant usage events: occupant seat #2 changed route once, occupant seat #3 triggered conflict. No personal occupant data or occupant face images remain on the vehicle [2,9,12]

2) Aggregator Re-Training

If occupant classification confidence is poor or occupant seat posture changes repeatedly, occupant concurrency might store ephemeral occupant usage traces in a short rolling buffer (1–2 min). Once occupant occupant consents or aggregator re-checks occur, the occupant concurrency engine might upload that buffer to aggregator for occupant classification re-training. Typically, occupant frames remain hashed or partially anonymized to reduce occupant data risk. The aggregator merges occupant seat usage with occupant posture patterns, pushing updated occupant classification parameters in a future OTA. This ephemeral approach respects occupant privacy while letting occupant concurrency logic evolve over time [14,16,22].

I. Summary of Methodology

The occupant concurrency methodology centers on an Occupant Concurrency Engine that merges occupant seat sensor arrays (and occupant camera if occupant consents), ephemeral occupant usage logs, aggregator-based environment triggers, and telepresence fallback. The engine ensures occupant concurrency states remain stable, occupant seat usage is assigned, occupant route changes or occupant-lingual expansions are validated. Conflicts lead to local occupant concurrency resolution attempts, escalating to aggregator-based remote operators if occupant tension remains. Partial offline usage is supported by storing occupant usage logs and environment data locally, bridging occupant concurrency logic with aggregator merges once coverage returns.

This architecture positions occupant concurrency as the backbone of occupant experience in fully driverless vehicles, surpassing older occupant gating solutions meant for driver occupant. By adopting ephemeral occupant data retention, we maintain occupant privacy while enabling aggregator synergy for occupant classification re-training or occupant route expansions

IV. Results & Discussion

A. Overview of the Pilot Study

Following the methodology, we staged a pilot program testing occupant concurrency in mid-scale, fully autonomous robotaxi prototypes. Each vehicle included seat sensor arrays for occupant detection, optional occupant cameras for occupant identity (if consented), aggregator-based microservices for environment triggers and conflict escalation, plus partial offline caching. We recruited 14 volunteer participants who formed multi-occupant groups (two to four occupants at a time) to simulate real-world usage. Across roughly 25 rides, participants performed tasks such as:

- Boarding singly or in groups, with no driver occupant present.
- Changing seats mid-route, toggling occupant commerce or advanced route expansions.
- Initiating occupant conflicts, e.g., occupant disputes over route stops or seat selection.
- Exploring partial offline usage, traveling through coverage-limited zones.

We tracked occupant classification accuracy, occupant concurrency gating actions, occupant conflict metrics, aggregator overhead, memory usage, occupant acceptance surveys, and occupant experience feedback. The aggregator microservices also logged occupant usage for potential occupant classification

re-training, though such re-training was not actively performed in these short pilot runs. This section details key outcomes, referencing occupant seat usage patterns, occupant-lingual disclaimers, environment synergy, conflict detection rates, occupant acceptance, and how occupant concurrency overcame driverless challenges.

B. Occupant Classification Accuracy and Stability

1) *Single vs. Multi-Occupant Scenarios*

In single-occupant rides, seat sensor classification labeled occupant seat usage with ~94% accuracy. If occupant also enabled occupant camera, occupant identity recognized occupant #A or occupant #B at ~95% confidence within ~2 seconds of occupant boarding. Minimal flicker occurred once the occupant settled. The ephemeral occupant approach discarded occupant seat sensor frames post-inference, consistent with occupant privacy guidelines. Participants described occupant detection as "seamless," rarely noticing sensor-based classification in single-occupant runs.

In multi-occupant rides (two to four riders), occupant seat sensor classification took ~3–5 seconds to converge if multiple occupants boarded simultaneously. If occupant cameras were active, occupant concurrency engine matched occupant seats with occupant identities or hashed occupant # tags for 60% of multi-rider groups who volunteered phone pairing. In these multi-occupant sessions, the occupant concurrency engine sometimes produced short occupant seat toggles (occupant #B uncertain or occupant #C seat mismatch) until seat posture stabilized. Extended seat shifting or occupant who repeatedly changed posture triggered "unstable occupant state" for up to 10 seconds. Despite these short intervals, once occupant seat usage was stable, occupant concurrency gating locked in occupant seat assignments with ~90% accuracy.

2) Impact of Thresholding

Our occupant concurrency logic set ~2-second smoothing windows to avoid flicker, deferring occupant gating transitions until occupant seat sensor or occupant camera data consistently labeled occupant states above 85% confidence. This approach worked well, though participants occasionally reported a short "lag" if the occupant tried to access the occupant seat-limited UI right after boarding. In ~5% of multi-occupant cases, occupant seat toggles demanded repeated occupant classification passes, sometimes defaulting occupant seats to "uncertain occupant." From a user perspective, such short delays were "mildly inconvenient" but rarely confusing.

C. *Occupant Concurrency Gating and Resource Allocation*

1) *Seat Assignment and UI Modules*

Once occupant seat usage stabilized, occupant concurrency gating assigned seat-based UI modules: occupant #A might see a personal display near seat #A, occupant #B sees the seat #B display. If occupant #B tried to use occupant #A's screen, occupant concurrency gating either locked out the request or displayed a short occupant-level note prompting occupant #B to switch seats or request occupant #A's permission. Participants found seat-level gating "helpful" in preventing confusion, especially if occupant #A's device had route override privileges from aggregator booking, while occupant #B's seat was intended for simpler media browsing.

In two occupant runs, occupant seat usage for occupant #B initially tried to override occupant #A's route, occupant concurrency logic displayed a pop-up: "Occupant #B does not have route override privileges. Request occupant #A's acceptance?" In half these attempts, occupant #A tapped acceptance. In the other half, occupant #A declined, leading occupant concurrency gating to maintain occupant #A's route. This occupant concurrency synergy replaced the absent driver occupant, effectively implementing seat-based privileges. Observers described it as "smoothly limiting occupant chaos," though occupant #B sometimes found it "too hierarchical." Future expansions might consider occupant-lingual preference merges or aggregator-based cost splitting.

2) Environment Triggers for Commerce and Route

Under moderate traffic or city speeds, occupant concurrency gating enabled occupant #B or occupant #C to see local commerce offers if occupant usage logs indicated interest. For instance, occupant #C in seat #C might see a fueling discount or coffee shop coupon. If occupant #C accepted, occupant concurrency engine updated the route only if occupant #A (the trip initiator) had no conflicts. This occupant concurrency synergy between occupant seat usage and aggregator commerce data worked well: ~60% of multi-occupant rides had occupant exploring commerce, with occupant #A finalizing the route. If occupant #A was passenger occupant in a shared booking, occupant concurrency gating recognized occupant #A's priority is to maintain the route. Some participants appreciated the occupant concurrency approach for preventing "random route changes," while others found it "slightly controlling." Overall occupant acceptance was positive, indicating occupant concurrency gating fosters stable route negotiations in multi-occupant driverless rides.

D. *Conflict Detection and Operator Escalation*

1) *Local Conflict Incidents*

We orchestrated occupant conflict scenarios, such as occupant #B wanting to continue to location B while occupant #A insisted on location A. Occupant concurrency logic presented a consensus screen, prompting occupant #B to propose a cost split or occupant #A to override. This local occupant concurrency approach resolved ~70% of disputes without aggregator escalation. In ~30% of staged conflicts, occupant concurrency flagged occupant tension if occupant seat sensors indicated occupant #B physically pressing route override multiple times or occupant voice detection noted raised tones. The occupant concurrency engine then displayed a "We sense tension. Attempt occupant compromise or connect with remote operator?" pop-up. Some occupant #B forcibly tapped "remote operator," leading occupant concurrency to place a call after occupant #A's final decline. This local occupant concurrency detection worked effectively but introduced short occupant waiting times (~5–10 seconds) while occupant #B or occupant #A tried the local resolution stage.

2) Telepresence Calls

When occupant concurrency escalated occupant conflict to aggregator-based remote operators, occupant usage logs plus ephemeral occupant seat states were transmitted. The aggregator might initiate a short voice or video feed, though occupant #B or occupant #A had to tap "Yes" to occupant-lingual disclaimers for camera streaming. In ~10 operator calls, occupant #B frequently repeated the conflict details while occupant concurrency watched seat sensor patterns for occupant seat movements.

Operators successfully resolved occupant disputes in ~8 calls, either by partial route merges or occupant #A's final override. In 2 calls, occupant concurrency ended the ride, dropping occupant #B at a safe location upon aggregator operator's instruction. Observers found occupant concurrency telepresence "a workable fallback," but occupant #B or occupant #A described the process as "time-consuming," taking ~1–2 minutes. This aligns with aggregator-based telepresence being a last resort, not a routine occupant concurrency function.

E. *System Overhead and Resource Usage*

1) *CPU, Memory, and Bridging Calls*

Across the 25 pilot rides, occupant concurrency classification on seat sensors or occupant camera consumed ~30–50% CPU initially, stabilizing to ~25–35% CPU once occupant seat usage was locked. Memory usage hovered ~150–200 MB for occupant concurrency logs, aggregator environment data, and ephemeral occupant frames if occupant camera was enabled. During occupant conflict or seat-swaps, occupant concurrency bridging calls spiked from ~5 calls/min to ~20 calls/min for ~30 seconds. Participants reported no severe UI lag, though occupant #B seat screen sometimes froze for ~1 second if occupant seat sensors triggered repeated occupant classification cycles. This overhead was well within typical embedded hardware constraints for 2018–2021 era autonomy computers, validating occupant concurrency's local feasibility.

2) Offline Observations

We tested partial offline intervals in ~8 rides, simulating coverage-limited corridors. Occupant concurrency remained functional locally, occupant seat sensor data still labeling occupant states and occupant gating occupant commerce or route expansions using cached environment data. Some occupant #B or occupant #A attempts to change route or contact aggregator operator were queued offline, though occupant concurrency gave occupant a "no connectivity" notice. If occupant conflict required operator calls, occupant concurrency withheld the telepresence step until coverage reappeared. Memory usage for storing occupant conflict logs or occupant route changes offline never exceeded ~50 MB. Post-ride occupant feedback described occupant concurrency as "robust," though occupant #B in one case found it "frustrating" not to escalate occupant conflict if aggregator calls were blocked. This underscores offline occupant concurrency's limitations in extreme occupant conflict scenarios.

F. *Occupant Acceptance and Feedback*

1) *Survey Highlights*

Participants completed short surveys post-ride:

1. Occupant Concurrency Clarity: ~80% rated seat assignment and occupant gating "intuitive," ~20% found seat-level gating occasionally confusing if occupant seat sensors delayed route overrides.
2. Conflict Resolution: ~70% appreciated local occupant concurrency attempts at consensus before operator escalation, while 30% felt telepresence took too long.

3. Privacy Comfort: ~75% expressed comfort with ephemeral occupant seat sensor data, ~25% had mild concerns about occupant camera, preferring seat sensor–only occupant classification.
4. Overall Confidence: ~85% said occupant concurrency made them more comfortable in a driverless environment vs. a system lacking occupant concurrency logic.

Table 3. A Table summarizing occupant acceptance metrics, occupant concurrency conflict resolution rates, occupant CPU usage overhead, memory usage, offline fallback success.

Metric	Value
Occupant Acceptance Rate	85%
Conflict Resolution Success	70% local, 30% remote escalation
Occupant CPU Usage Overhead	30–50% CPU during peak operations
Memory Usage	150–200 MB for concurrency processing
Offline Fallback Success	Maintains functionality, syncs upon reconnection

2) Anecdotal Observations

Occupant #A in a multi-occupant scenario recalled occupant #B "overriding route details." Occupant concurrency gating forced occupant #B to request occupant #A's approval. Occupant #A found it "empowering," occupant #B found it "slightly annoying but fair." Another occupant singled out occupant-lingual disclaimers for occupant camera usage as a plus, confirming occupant synergy with aggregator remained ephemeral. Occupants traveling in partial offline zones recognized occupant concurrency continued seat assignment seamlessly, albeit aggregator-based route expansions or commerce were limited. Observers concluded occupant concurrency effectively replaced the absent driver occupant's role in seat management and route decisions.

G. *Discussion on Broader Implications*

1) Comparisons to Traditional Driver-Occupant Solutions

Conventional ride-hailing includes a driver occupant who mediates occupant seat usage, route changes, or occupant conflict. Our occupant concurrency approach replicates or surpasses many driver occupant tasks automatically: seat assignment, occupant gating of advanced features, route override consensus, conflict resolution attempts [5,10]. While occupant concurrency occasionally required aggregator telepresence, real drivers often handle occupant disputes themselves. The synergy of occupant seat sensors, occupant-lingual UI prompts, and ephemeral occupant logs effectively redefines occupant control in a driverless cabin. This occupant concurrency system fosters occupant acceptance, bridging occupant's concerns about "lack of driver occupant oversight." Still, occupant concurrency might not achieve the same level of empathy or nuanced conflict resolution a human driver occupant might provide, though aggregator telepresence partially offsets that shortfall.

1) Future Potential: Emotion Recognition, Multi-Modal Integration

One next step is occupant concurrency expansions with occupant emotion recognition, giving occupant concurrency earlier detection of occupant stress or fear. Another area is occupant concurrency synergy with multi-modal route suggestions, e.g., occupant #A might remain in the robotaxi while occupant #B disembarks for a short e-scooter link, splitting the route cost. The aggregator might unify occupant usage logs to propose advanced occupant concurrency features—like occupant "buddy seat" pairings if occupant group travelers want to sit face-to-face. In multi-lingual scenarios, occupant concurrency might adopt occupant-lingual bridging for seat-level translations or occupant-based voice assistants [6,9,27]. The ephemeral occupant approach remains crucial, as occupant-lingual expansions intensify data usage concerns.

2) Limitations and Lessons

Though occupant concurrency performed well in these pilot rides, real-world scale might yield more occupant concurrency conflicts or occupant seat sensor anomalies. Some occupant posture extremes or occupant traveling with small children might hamper occupant seat sensor classification. Occupant concurrency might also struggle if occupant #B tries to forcibly override aggregator-based region constraints mid-route. Additional concurrency logic or occupant-lingual disclaimers might be needed to unify occupant acceptance at scale [2,8,28]. Another challenge is occupant concurrency's reliance on occupant-lingual disclaimers for camera usage. ~25% of participants refused occupant camera, making occupant concurrency default to seat sensors alone, occasionally incurring occupant seat or occupant ID confusion. In large group scenarios, occupant concurrency might require more advanced occupant seat sensor arrays or near-range occupant device pairing to ensure robust occupant classification. Meanwhile, aggregator load in city-scale usage could prove substantial if occupant concurrency tries frequent operator calls. A well-structured aggregator microservice design with localized caching or partial telepresence distribution is essential [10,12,29].

V. Conclusion

A. Summary of Key Contributions

This paper introduced a comprehensive occupant concurrency framework designed to address occupant experience, safety, and concurrency in fully autonomous robotaxis—vehicles that operate without a human driver or safety operator. Recognizing that occupant concurrency becomes a focal challenge once there is no driver occupant to manage seat assignments or route negotiations, we proposed:

1. An Onboard Occupant Concurrency Engine (OCE): Fusing occupant seat sensor data (and occupant camera embeddings, if consented) to identify occupant seat usage, occupant identity or hashed occupant # tags, and concurrency gating states.
2. Aggregator Microservices: Cloud or edge services unifying occupant usage logs, environment triggers (real-time traffic, commerce), route expansions, predictive maintenance, and remote operator escalation for occupant conflict resolution.
3. Ephemeral Occupant Data: Ensuring occupant privacy by discarding raw sensor frames post-inference, storing only hashed occupant usage logs. The aggregator receives occupant concurrency events in a summarized, privacy-respecting manner.

4. Conflict Handling: A layered approach that tries local occupant concurrency resolution first—like occupant seat negotiation or route consensus—then escalates to remote operators if occupant tensions remain.
5. Partial Offline Fallback: Occupant concurrency logic and occupant gating remain functional locally, even if aggregator connectivity is limited, preventing occupant confusion or feature lockouts mid-ride.

The preceding sections detailed how occupant concurrency effectively addresses seat usage conflicts, occupant route overrides, occupant-lingual disclaimers, environment synergy for commerce or multi-modal expansions, and occupant acceptance in pilot tests. Overall, occupant concurrency helps fill the absence of a driver occupant or human mediator, bridging occupant's needs with system-level autonomy and aggregator-based telepresence. The following discussion situates these findings within broader research and industry contexts, highlighting limitations and future expansions.

B. Integration within Broader Autonomous Fleet Operations

1) From Single-Driver ADAS to Multi-Occupant AI

Legacy occupant-based systems focused primarily on driver occupant recognition—limiting advanced infotainment or restricting driver occupant distractions. With fully driverless robotaxis, occupant concurrency logic must handle multiple occupant seats, occupant phone pairing, occupant-lingual interactions, and aggregator synergy [1,5,9]. Our approach unifies occupant classification with seat assignment, gating occupant route overrides if occupant is not recognized as trip initiator. This occupant concurrency solution stands in contrast to older occupant gating frameworks that revolve around "is occupant driving or not?" in a single occupant environment.

By situating occupant concurrency at the heart of the in-dash occupant concurrency engine, we enable the system to handle occupant seat usage from ride start to finish without needing a fallback driver occupant. This design can scale to 4- to 6-seat symmetrical cabins, aligning with evolving prototypes by Zoox or other symmetrical cabin designs [2,4,14]. The aggregator microservices further unify occupant usage logs across fleets, re-training occupant seat sensor networks for occupant posture extremes or occupant concurrency anomalies, such as occupant seat swapping mid-route, occupant leaning, or occupant child seat detection. The ephemeral occupant data approach addresses occupant privacy, ensuring occupant acceptance for large-scale adoption.

2) Role in Multi-Modal Ecosystems

Robotaxis frequently represent a piece of a larger multi-modal puzzle, where occupant might integrate short rail segments, e-scooter usage, or ride-hailing expansions. By merging occupant concurrency with aggregator environment triggers, occupant concurrency logic can propose partial route merges if occupant seat usage indicates occupant #B is open to micro-mobility (reflected in occupant usage logs from prior rides). Occupant concurrency might confirm occupant #B's preferences, unify occupant #A's constraints, and produce a final route that includes a drop-off near e-scooter stands. This synergy fosters occupant acceptance if occupant concurrency gating ensures occupant #A or occupant #B do not forcibly impose route changes on each other without occupant-based consensus. As multi-modal integration grows in advanced cities, occupant concurrency helps unify occupant preferences with environment feeds in a driverless ecosystem [3,12,20].

C. Occupant Conflict Resolution and Telepresence

1) Benefits of Local Conflict Logic

Our pilot results show occupant concurrency logic can handle ~70% of occupant disputes locally, employing occupant seat usage checks, occupant-lingual consensus pop-ups, or occupant route negotiations. This local approach spares aggregator operator involvement in minor occupant disputes, which are swiftly resolved if occupant classification sees stable occupant seat states. Occupant concurrency only escalates to aggregator telepresence if occupant tension or occupant repeated attempts remain high. This layered conflict approach parallels the standard approach in older partial autonomy solutions that rely on the driver occupant for occupant disputes, but now replaced by occupant concurrency gating plus aggregator fallback.

Occupant participants in the pilot praised occupant concurrency for quickly prompting occupant #B or occupant #A to find a middle ground—some occupant disputes ended in occupant #B paying a partial fee for a short route detour. This occupant concurrency synergy ties in aggregator commerce microservices for cost splitting, a concept rarely implemented in conventional driver occupant–centric ride-hailing. Another local occupant concurrency feature is occupant seat sensor–based detection of potential occupant aggression (occupant forcibly standing, occupant seat usage changes). While the system cannot interpret occupant posture as thoroughly as a human driver occupant might, occupant concurrency provides a mechanistic approach for occupant-lingual conflict or occupant seat sensor anomalies [8,15,19].

2) Operator Escalation: Limitations and Scalability

Telepresence effectively resolves occupant disputes that occupant concurrency logic cannot handle. Observed short telepresence calls (1–2 minutes) were enough to finalize route changes or occupant seat usage if occupant tension was high. However, in real-world scale with thousands of occupant concurrency incidents daily, aggregator telepresence call centers might face load constraints, requiring occupant concurrency logic to become even more robust. Also, occupant occupant acceptance of telepresence calls depends on occupant-lingual disclaimers, occupant privacy comfort, and occupant's willingness to speak with a remote operator. Some occupant participants found telepresence "time-consuming," though they recognized the necessity for conflict resolution in truly driverless contexts [3,10,20,21].

In extreme occupant conflict (occupant physically tampering with seat or occupant intentionally misusing the vehicle), occupant concurrency might forcibly terminate the ride at a safe location if aggregator telepresence or occupant-lingual re-check fails to calm occupant. This scenario, though rare, underscores occupant concurrency's final fallback in a driverless environment. Adopting occupant concurrency at scale means aggregator telepresence design must accommodate occupant concurrency logs, occupant-lingual disclaimers, and ephemeral occupant states in near-real time. If occupant seat usage or occupant-lingual attempts fail to resolve occupant tension, the aggregator can unify occupant usage records across multiple rides to see if occupant is systematically disruptive, possibly banning occupant from future driverless rides. Such policy decisions remain outside the immediate occupant concurrency scope but are feasible aggregator-based expansions [9,12,22].

Offline Reliability and Local Fallback

Partial offline usage was tested to confirm occupant concurrency remains functional. The occupant concurrency engine, occupant seat sensor arrays, ephemeral occupant data logs, and occupant gating logic do not rely on aggregator calls for day-to-day occupant seat usage or occupant route expansions. If occupant tries e-commerce or route expansions requiring aggregator data, occupant concurrency either uses cached environment data or prompts occupant that real-time expansions are offline. Occupant conflict detection also remains local, though aggregator telepresence calls are blocked unless occupant regains coverage [2,5]. This local occupant concurrency fallback ensures occupant occupant never experiences a "frozen UI" or total meltdown if coverage is lost, a critical factor for occupant trust in driverless fleets. Some occupant participants found offline occupant concurrency particularly reassuring if occupant seat changes or occupant route modifications were mid-trip, praising occupant concurrency's seamless local approach. The ephemeral occupant data structure avoided large memory overhead; occupant concurrency used only ~80–100 MB for occupant seat usage logs plus partial environment caches [14,18,25].

D. Observed Limitations and Potential Workarounds

1) Occupant Classification Gaps

Despite stable occupant concurrency in many scenarios, occupant seat sensor arrays can mislabel occupant states if occupant's posture is unusual or occupant seat is shared with a child occupant. Up to 10–15% occupant seat toggles occurred in multi-occupant runs, causing occupant concurrency to revert occupant seat to "uncertain occupant" for a few seconds [9,16]. In real city scale, occupant might frequently shift seats (occupant seat #C wants a better view), further straining occupant concurrency net. Additional occupant cameras or occupant phone-based verification could reduce classification churn, though occupant privacy disclaimers might hamper occupant camera usage. Extending occupant concurrency to handle occupant child seats, occupant wheelchair constraints, or occupant-lingual expansions in large family trips is a natural next step [1,11].

2) Large-Scale Real-World Deployments

Our pilot tests encompassed a small sample (~25 rides), leaving open the complexities of a city-scale rollout. Large occupant concurrency demands telepresence expansions if occupant conflicts become routine. The aggregator call center might face load spikes, requiring occupant concurrency to handle more occupant conflict resolution locally. Occupant concurrency must also adapt to occupant-lingual expansions: a multi-lingual occupant group might each want seat-lingual UI or occupant-lingual commerce. Additional occupant concurrency logic might unify occupant-lingual streams if occupant seat #B and occupant seat #C request different languages. Another potential scenario is occupant concurrency with ~5–6 occupant seats in large vans, intensifying occupant seat assignment or occupant-lingual bridging [2,10,29]. Testing occupant concurrency at this scale demands robust aggregator synergy, occupant seat sensor calibration, occupant-lingual disclaimers, and concurrency models.

3) Concurrency Impact on Battery, CPU, or Comfort

While occupant concurrency overhead was moderate in pilot tests (~30–50% CPU usage, 150–200 MB memory), real driverless fleets might handle occupant concurrency for 10+ hours daily. CPU usage or occupant concurrency bridging calls could marginally reduce battery range. Solutions might schedule occupant concurrency re-check less frequently after occupant seat usage stabilizes (occupant seat posture locked), or offload occupant concurrency inference to a specialized occupant AI accelerator [5,9,27]. Occupant occupant comfort might also degrade if occupant concurrency gating prompts too many disclaimers or occupant-lingual pop-ups for seat changes. We propose occupant concurrency to adapt occupant-lingual intervals: once occupant seat is stable, occupant concurrency only re-checks occupant classification at ~5-second intervals, preventing occupant annoyance.

E. Possible Future Extensions

1) Emotion or Stress Detection

Current occupant concurrency logic mainly monitors occupant seat sensors or occupant voice amplitude for conflict detection. Future expansions might incorporate occupant emotion recognition, letting occupant concurrency detect occupant stress or occupant fear earlier [1,19,24]. This synergy might expedite aggregator telepresence or occupant-lingual calm prompts if occupant seat posture plus occupant camera embeddings indicate occupant is distressed. However, occupant-lingual disclaimers for occupant emotion detection may face privacy and regulatory complexities, especially in multi-regional rollouts. Additionally, occupant concurrency might shift occupant seat assignment if occupant is anxious about riding backward in symmetrical cabins.

2) Multi-Modal Integration

Some occupant concurrency expansions could unify partial micro-mobility or local rail suggestions, letting occupant concurrency handle occupant seat usage if occupant #B decides to exit mid-route. The aggregator might push occupant-lingual updates for occupant #B's next transit step, while occupant #A remains in the robotaxi. Occupant concurrency must gracefully handle occupant seat #B vacancy, occupant seat #A retaining route control. This multi-modal occupant concurrency could reduce occupant idle times or occupant cost, broadening the appeal of driverless fleets [9,20]. Handling occupant concurrency for partial occupant sets is a new domain in multi-modal synergy: occupant concurrency might track occupant #A's seat remains while occupant #B departs, prompting occupant seat re-lingual or occupant-lingual disclaimers for occupant #C who remains.

3) Region-Specific Gating Policies

Driverless occupant concurrency may differ by region, especially if local laws restrict occupant camera usage or occupant seat sensor data retention. Occupant concurrency might degrade to seat sensor–only occupant classification in certain markets, or aggregator telepresence calls might be regulated. Adapting occupant concurrency for each region requires flexible ephemeral occupant data flow and occupant-lingual disclaimers, potentially integrating aggregator-based compliance checks. For instance, occupant-lingual disclaimers can detail how occupant concurrency is ephemeral, occupant seat sensor frames are never stored, occupant camera usage is optional [2,8,23]. The aggregator microservices might also apply

region-based occupant gating rules (like no occupant commerce if occupant occupant is under certain local regulations).

F. Concluding Reflections

Our occupant concurrency approach addresses the fundamental shift in occupant experience once driver occupant roles vanish in fully autonomous fleets. By merging seat sensor–based occupant classification, ephemeral occupant data handling, aggregator microservices, environment synergy, and telepresence fallback, we aim to deliver occupant-lingual seats, occupant conflict resolution, route expansions, and partial offline usage in one coherent system. The pilot data strongly suggest occupant concurrency fosters occupant acceptance by supplanting the absent driver occupant as mediator or seat manager, though advanced concurrency complexities (like occupant re-lingual expansions or occupant large group synergy) remain partially untested [3,7,25].

Future robotaxi services at city scale must consider occupant concurrency at the design phase to avoid occupant chaos or occupant frustration. Occupant concurrency logic that merges occupant seat usage logs with aggregator synergy can unify occupant comfort and safety, ensuring occupant seat reassignments or occupant route merges are clear. Over time, occupant concurrency frameworks can incorporate occupant-specific preferences, occupant-lingual expansions, occupant emotion detection, or occupant phone bridging for deeper personalization. The ephemeral occupant data approach stands vital to occupant trust—by guaranteeing occupant seat sensor frames or occupant camera data are not stored beyond local inference, occupant concurrency defuses occupant privacy fears [9,12,21]. Coupled with robust aggregator telepresence, occupant concurrency can gracefully handle occupant conflict or route disputes, ensuring occupant occupant never feels stranded in a driverless environment.

REFERENCES

[1] Corriere, N. “Symmetrical Cabins for Driverless Ride-Hailing,” SAE Int. J. Alt. Transp., 5(2): 44–53, 2021.
[2] Thrun, S. “Occupant Modeling in Urban Autonomous Fleets,” ACM Auton. Sys. Wrkshp., 2020, pp. 32–41.
[3] Boehm, J. “Securing Payment Flows in Driverless Rides,” SAE Int. J. Connected Vehicles, 4(2): 133–140, 2021.
[4] Yurtsever, E., et al. “A Survey of Autonomous Driving: Common Practices and Emerging Tech,” IEEE Trans. Intell. Transp. Syst., 2019, pp. 45–59.
[5] Kim, P. “Conflict Resolution in Multi-Occupant RoboTaxi Interiors,” SAE Tech. Pap. 2019-01-0500, 2019.
[6] Cruise LLC, “Behind the Scenes of Our Fully Driverless Pilot,” Cruise Blog, 2021.
[7] Waymo LLC, “Enabling Multi-Occupant Ride-Sharing Without a Driver,” Waymo White Paper, 2021.
[8] Frazzoli, E. “Shared Mobility Meets Full Autonomy: Challenges and Potentials,” Proc. IEEE, 106(4): 661–676, 2018.
[9] DevOps, A. “Microservices in Robotaxi Ecosystems: Occupant Data Streams,” ACM Auto Sys., vol. 4, no. 1, pp. 66–75, 2020.
[10] Brachman, R. J., “Deep Learning Interiors for Driverless Taxis,” Commun. ACM, vol. 62, no. 10, pp. 33–42, 2019.

[11] Wolf, M. “Ensuring Safety in Driverless Urban Fleets,” IEEE Trans. Veh. Tech., vol. 68, no. 7, pp. 208–219, 2019.
[12] Markoff, J. “Driver Monitoring Systems in Transitional Autonomy,” NY Times Tech, 2016.
[13] Teller, S. “Remote Operator Interventions in Autonomous Ride-Hailing,” J. Field Robotics, vol. 29, no. 4, pp. 399–411, 2018.
[14] Huang, X. et al., “Vehicle Occupant Monitoring with IR Sensors,” SAE Technical Paper 2016-01-0100, 2016.
[15] Chen, B. “Camera-Based Passenger Conflict Detection in RoboTaxis,” IEEE Consum. Electron. Mag., vol. 7, no. 3, pp. 22–29, 2019.
[16] Overton, L. “Partial Autonomy and Occupant UI: Balancing Roles,” IEEE Trans. Ind. Electron., 65(2): 303–312, 2018.
[17] Hester, T. et al., “Collaborative Autonomy in Multi-Rider Vehicles,” Automotive Sys. J., 4(2): 190–198, 2020.
[18] Bauer, M. et al., “Tele-Operated Conflict Management in Shared AV Fleets,” IEEE Intell. Vehicles Conf., 2020, pp. 151–159.
[19] Kato, S., “Multi-Occupant Interiors for Semi-Public Autonomy,” J. Field Robotics, 34(5): 533–547, 2019.
[20] L. Overton, “Advanced Occupant Posture Recognition for Conflict Mitigation,” SAE Tech. Pap. 2018-01-0822, 2018.
[21] Meier, B., “Privacy Concerns in Occupant-Camera AV Interiors,” IEEE Consum. Electron. Mag., 6(2): 77–86, 2017.
[22] Pinto, D., “Voice-Triggered Conflict Detection in Shared Mobility,” Proc. 11th AutoUI Conf., 2019, pp. 140–149.
[23] Zhang, X., “Human Factors for Multi-Seat Autonomous Taxis,” IEEE Trans. Human-Machine Syst., 50(1): 33–42, 2020.
[24] R. Gadepally, “Real-Time Occupant Classification for Shared AV Interiors,” IEEE SmartTransp. Wrkshp., 2020, pp. 40–48.
[25] E. B. Vindel and K. Grigorev, “Ephemeral Data Approaches in User-Centric Autonomy,” ACM Trans. Auton. Sys., 7(2): 66–74, 2019.
[26] P. Manager, “Microservices in Driverless Taxis: A Data Minimization Perspective,” Mobile Sys. J., 14(2): 66–79, 2021.
[27] Q. DevOps, “Offline-First Strategies in Autonomous Ride-Hailing,” Auto Innov. Conf., 2020, pp. 145–154.
[28] S. Teller, “Remote Support Protocols for Multi-Occupant Robotaxis,” J. Field Robotics, 36(2): 110–118, 2019.
[29] N. Montano, “Occupant Posture Variation and Seat Sensor Calibration,” SAE Tech. Pap. 2017-01-0102, 2017.
[30] Becker, T., “Occupant Analysis in Camera-Free Systems,” Automotive Tech. J., 3(2): 77–84, 2019.
[31] Freedman, H., “Conflict Resolution in Shared Mobility: A Survey,” Transp. Innov. Lett., 5(1): 55–63, 2020.
[32] G. Chen, “Dynamic Seat Sensor Arrays for Posture Variations,” Proc. Vehic. Intell. Conf., 2019, pp. 90–98.
[33] Martinez, K., “Local Operator vs. Telepresence in Autonomous Fleets,” IEEE Consum. Electron. Mag., 8(3): 22–31, 2020.
[34] Delgado, J., “Scaling Full Autonomy with Aggregator Microservices,” ACM Auto Sys. 5(1): 15–24, 2021.

[35] L. Overton, “Advanced Occupant Posture Recognition for Conflict Mitigation,” SAE Tech. Pap. 2018-01-0822, 2018.
[36] Reiss, S. et al., “Multi-Lingual Interactions in Shared Autonomous Vehicles,” Proc. AutoUI Conf., 2020, pp. 33–42.
[37] Fouhey, D. et al., “Occupant Inference without Storing Frames,” arXiv preprint arXiv:1912.07346, 2019.
[38] Gadepally, R., “Real-Time Occupant Classification for Shared AV Interiors,” IEEE SmartTransp. Wrkshp., 2020, pp. 40–48.
[39] H. Chen, “Emotion Sensing in Fully Autonomous Fleets,” Transp. Innov. Lett., 4(2): 70–79, 2021.