Software Updates for Large-Scale Autonomous Vehicles Deployment: Challenges and Opportunities

Arpan Bhattacharjee and Weisong Shi, *University of Delaware, Newark, DE 19713, USA*

As self-driving vehicles transition from experimental prototypes to commercial fleets, they rely on increasingly complex software that must evolve regularly to stay safe and competitive. This article examines the primary challenges of software over-the-air (OTA) updates for large-scale autonomous vehicle (AV) deployments and outlines the technical and operational frameworks for managing these challenges. We argue that OTA capability is not optional but essential for fleet safety and competitiveness. Nevertheless, the complexity of AV code bases, the fragility of tightly coupled systems, the logistics of phased rollouts, network limitations, and cybersecurity threats all complicate updates. By posing key questions that engineers and operators must address—and by grounding the discussion in recent incidents involving Waymo and Zoox—we demonstrate how careful architecture, simulation-driven validation, robust security, and scalable infrastructure can transform OTA into a reliable pillar of AV operations. We conclude with research directions for the next decade of OTA development.

Autonomous vehicles (AVs) are ultimately software-defined machines. AV's perception, prediction, and control modules depend on artificial intelligence (AI) algorithms that learn from data and improve over time. High-definition maps, sensor calibrations, and vehicle diagnostics must evolve in tandem to ensure seamless integration. When a bug is discovered or a new capability is ready, it is not feasible to recall hundreds of vehicles to a dealership or wait for annual servicing. Fleets need to deploy a fix or improvement quickly so that they can avoid the risk of accidents, reputational harm, and regulatory penalties. Figure 1 illustrates the transition from traditional, dealership-centered software updates to cloud-enabled over-the-air (OTA) updates, highlighting how autonomous vehicles increasingly rely on continuous, wireless delivery rather than manual, offline methods.

Recent events illustrate the stakes. In 2025, Waymo recalled more than 1200 of its self-driving taxis after discovering that its software could mis-score stationary objects, such as chains and gates, leading to low-speed collisions.[1] The fix was rolled out across the fleet in weeks via software OTA update, demonstrating the necessity of OTA updates. Similarly, Amazon's Zoox halted its robotaxi operations in one city after an unoccupied shuttle collided with another vehicle; an OTA patch allowed service to resume.[2] These examples demonstrate that OTA updates are crucial for maintaining safety and trust, but must be executed meticulously to prevent introducing new hazards.

The rise of OTA also stems from the extraordinary complexity of modern vehicle software. A connected car may contain 80–100 million lines of code, and fully self-driving prototypes could exceed 250–300 million lines.[3] This dwarfs the software of smartphones or even aircraft. Additionally, AV fleets operate under varying conditions and jurisdictions; therefore, the software

1089-7801 © 2025 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies.
Digital Object Identifier 10.1109/MIC.2025.3643221
Date of current version 23 January 2026.

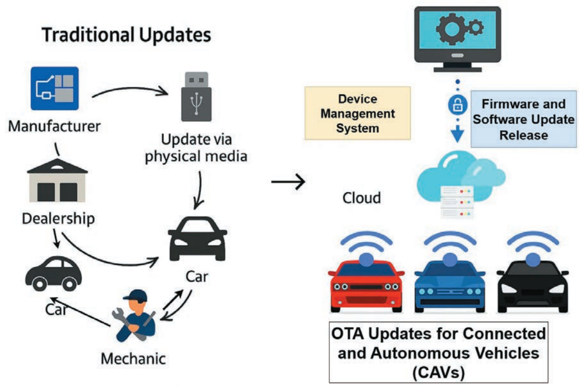
Traditional Updates
Manufacturer
Update via physical media
Dealership
Car
Car
Mechanic
Device Management System
Firmware and Software Update Release
Cloud
OTA Updates for Connected and Autonomous Vehicles (CAVs)

FIGURE 1. Evolution of in-vehicle software updates: Traditional dealer-based updates relying on physical media and mechanics contrasted with modern OTA workflows for autonomous vehicles, where updates are delivered wirelessly through cloud infrastructure.

must adapt to different road rules, sensor suites, and hardware revisions. The challenge, thus, is to update an entire distributed system without breaking it. Each section of this article addresses a challenge framed as a question that engineers and operators must answer to deliver trustworthy OTA updates at scale. The discussion draws on industry reports, academic research, and real-world incidents to illustrate how these questions can be answered in practice.

CAN WE MANAGE THE SCALE AND COMPLEXITY OF AV SOFTWARE?

One might ask whether it is even possible to manage software updates when a single vehicle runs on hundreds of millions of lines of code and dozens of subsystems. In a self-driving stack, perception modules process high-dimensional sensor data from lidar, radar, and cameras; prediction models estimate the behavior of nearby actors; planning algorithms determine trajectories; and control software actuates steering, braking, and acceleration.[4] Different teams or suppliers may develop each module and may depend on distinct hardware accelerators. A minor change in one module can ripple across the system; for example, an update to the neural network that classifies pedestrians might alter the timing of messages passed to the planner, which in turn could change the vehicle's trajectory. The risk is that a minor bug in one part of the stack could manifest as a safety-critical failure elsewhere.

Why Not Update Infrequently?

Fewer, larger updates lead to less disruption. However, in the context of AV, the stakes are reversed: Long delays become a risk. Accumulating many changes into one significant release makes it harder to isolate regressions and delays the deployment of safety fixes. The better strategy is to adopt a continuous-delivery approach, shipping more minor, well-scoped updates that can be validated and rolled back with minimal blast radius. Waymo's response to its towed-truck incident illustrates this philosophy: The company quickly developed a patch and began deploying it within days. Frequent updates enable engineers to address edge cases as soon as they are discovered, thereby reducing the number of unknown interactions per release.

How Can We Validate Updates for Safety?

Thorough validation is indispensable. Unlike a mobile app, an AV software update must demonstrate that it does not degrade safety in any foreseeable scenario. Modern operators use multistage pipelines: Unit tests and static analysis catch simple errors; replay tests run recorded sensor data through the new software to compare outputs; large-scale simulation exercises the software across millions of synthetic scenarios, including rare events and adversarial edge cases; and shadow mode deployments allow new software to run passively on vehicles without controlling them. This layered approach acknowledges that simulation alone cannot prove safety, but it can uncover many issues before on-road testing. After simulation and track tests, updates are deployed to a small cohort of vehicles—sometimes referred to as a canary deployment or Ring-0—whose telemetry is closely monitored. Only when no anomalies appear does the rollout expand to larger cohorts. This staged approach helps manage complexity and minimize risk while enabling rapid improvement.

How Fragile Are Distributed Vehicle Systems During Updates?

Another question concerns the interdependence of the vehicle's components. A modern car is a network on wheels that consists of dozens of electronic control units (ECUs), domain controllers, sensors, and actuators. In an autonomous vehicle, this network is further complicated by the use of high-performance computing platforms for AI processing. Updating just one ECU without synchronizing its dependencies can cause mismatches. For example, the braking system may expect messages in a different format from the updated planning module, leading to errors. This fragility arises because components often have implicit timing and interface contracts that are not explicitly defined.[5]

To address this, OEMs are moving toward coordinated, complete system updates, where a central orchestrator manages versions and dependencies across multiple ECUs. Instead of flashing one module at a time, the update package includes all the necessary firmware and software for the targeted configuration, ensuring that interdependent modules evolve in a coordinated manner. If any component fails to update, the orchestrator can reroll the entire transaction, preventing vehicles from entering inconsistent states.[6] Hardware heterogeneity further complicates matters: Fleets contain vehicles of different model years, sensor suites, and computing platforms, so an OTA system must tailor the update package to the specific build of each car. Industry analysts note that heterogeneity "creates a significant challenge in designing OTA updates compatible with a diverse fleet." Therefore, robust configuration management and version tracking are critical; they ensure that each vehicle receives the correct package for its hardware and that operators can audit which version is running on which vehicle at any time.

Managing OTA for End-to-End Multimodal Models for AV

End-to-end models are large and powerful, but a single mistake can affect the whole stack. Treat the model like a product: ship weights, pre- and post-processing, and calibrations as a single signed package; keep an A/B copy for instant rollback; and shrink downloads with deltas, chunking, and quantization. Turn the model on gradually—by city, weather, or time of day—and require Canary metrics to check first. While it runs, watch for distribution shifts and safety envelope breaches, and revert to a conservative controller if anything appears amiss. Run the new model in shadow against the old one beforehand, then evaluate it based on clear measures, including safety events, rule compliance, latency/power, and rollback rate. If those numbers don't hold, don't scale it to the fleet.

HOW SHOULD UPDATES BE ROLLED OUT ACROSS LARGE FLEETS?

Performing an update to thousands of vehicles simultaneously is risky. If a defect passes through, it could affect an entire service area at once. Instead, fleet operators should implement phased rollouts, where an update is first sent to a small, representative subset of vehicles in a controlled environment. Telemetry from this cohort will be monitored for anomalies such as unexpected braking events or system reboots. If the software behaves as expected, the rollout expands gradually—from a few vehicles to a city, then to multiple cities. This ring-based deployment aligns with practices in web services, ensuring that any issues can be detected before reaching a large customer base. Data-driven decision making is crucial: Automated monitors can detect statistically significant deviations in safety metrics between updated and nonupdated vehicles, triggering a halt or rollback.

Operational logistics also matters. Vehicles cannot receive critical updates while in motion or carrying passengers; updates may require the car to be parked with a sufficient battery. Fleet operators schedule updates during low-demand periods or when vehicles return to the depots for charging and cleaning. This means that large fleets can be updated over several days or weeks, depending on the availability of vehicles and network capacity. Operators must avoid leaving vehicles with outdated software for an extended period, as this could expose them to both known and unknown defects. A robust OTA system tracks the status of the update and retries downloads or installations for vehicles that miss the first attempt.

WHY ARE PARTIAL INSTALLS UNSAFE?

Autonomous vehicle updates rarely affect just one artifact; new features or fixes span code, high-definition (HD) maps, calibrations, models, and driving policies. Updating any of these in isolation risks mismatches; for example, a planner expecting new map semantics

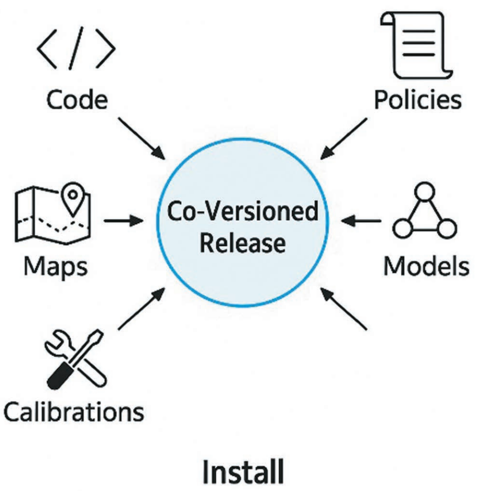

FIGURE 2. Conceptual illustration of the co-versioning principle: All interdependent artifacts—code, maps, calibrations, models, and policies—are tied to a single release and installed as one transaction.

while the car still carries an old pack, leading to degraded performance or unsafe behavior. Figure 2 illustrates the principle: All interdependent artifacts are tied to a single release and installed as one transaction, ensuring stack consistency by design.

How the Transaction Works

Before activation, the vehicle resolves a compatible set of artifacts for its hardware, verifies signatures, checks power/storage, and runs quick install gates (interface/latency checks, schema matches, confidence/accuracy thresholds for models, and basic map/region safety checks). Only if all gates pass does the system flip feature flags for the canary cohort. If any gate fails or if post-activation telemetry exceeds a threshold, the vehicle automatically returns to the previous software version.

ARE WIRELESS NETWORKS AND BACK-END INFRASTRUCTURE ROBUST ENOUGH?

OTA updates are based on network connectivity and cloud infrastructure. An AV software package may include new AI models, high-definition map tiles, or firmware images, generating gigabytes of data. Transmitting these packages over 4G/5G networks to hundreds of vehicles can strain bandwidth. Connectivity varies: Some cars park underground or in rural areas with poor coverage. Long downloads increase the risk of corruption or power loss, which could leave the vehicle in an inconsistent state. To mitigate these issues, OTA systems utilize delta updates, as shown in Figure 3, which transmit only the differences between the new and old software, thereby reducing file sizes and download times.[7] Updates are downloaded in the background and installed only when the vehicle is parked safely and has an adequate battery. The resume and rollback capabilities ensure that if a connection drops mid-download or mid-install, the process can restart without breaking the ECU. Edge caching and content delivery networks help scale updates. Files can be mirrored on servers closer to vehicles to reduce latency. Some approaches propose peer-to-peer dissemination for noncritical data, where vehicles share map updates or significant static assets with nearby peers using Wi-Fi or vehicle-to-vehicle links; critical firmware updates would still come from authenticated servers. Regardless of the distribution strategy, update campaigns must be orchestrated to avoid saturating the network. Operators may schedule downloads at off-peak times or throttle the number of concurrent downloads per cell tower. These techniques ensure that vehicles remain connected for safety telemetry even during large update campaigns.

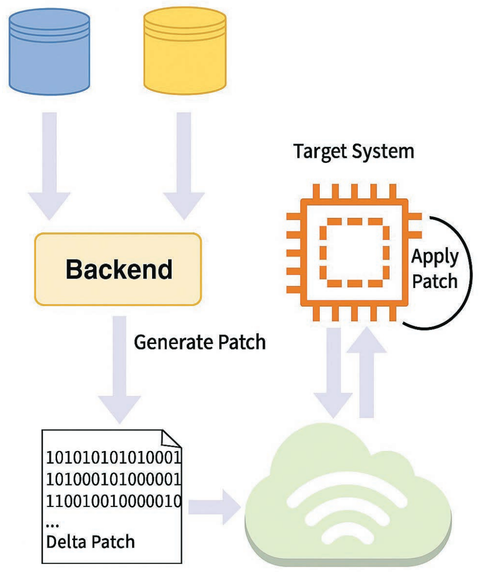

FIGURE 3. OTA update workflow: The back-end generates the patch, and the runtime system receives and applies it through the CAV connectivity infrastructure.

How Do We Secure the OTA Pipeline?

Because OTA systems extend the OEM's software supply chain into every vehicle, they present an attractive attack surface for cybercriminals. An attacker who could inject malicious code or intercept updates could gain control over vehicle functions or extract sensitive data. Past demonstrations, such as the famous Jeep Cherokee hack, illustrate the real danger: Researchers remotely exploited the vehicle's infotainment system to control steering and braking. Thus, every step in the OTA process must be secured. The manufacturer must digitally sign the update packages, and the vehicle must verify the signatures before installation. Transmission channels should use strong encryption to prevent eavesdropping or tampering. Many vehicles embed hardware security modules (HSMs) or trusted platform modules that securely store cryptographic keys and perform signature verification.[8]

Modern OTA frameworks also include secure boot and A/B partitions. Secure boot ensures that ECUs only run firmware that has been authenticated and verified. A/B partitioning means that updates are written to an inactive partition while the active software continues to run; once verified, the system switches to the active partition. If something fails, the ECU can revert to the previous working version. Beyond these mechanisms, proactive intrusion detection monitors network traffic and system behavior for anomalies, and incident response plans enable the fleet to quickly revoke or roll back updates if a vulnerability is discovered. These measures significantly increase the cost for attackers and ensure that OTA remains a security asset rather than a liability.

Do Infrastructure Dependencies Limit Update Deployment?

Autonomous fleets rely on a broader ecosystem that extends the vehicle and the cloud. Connectivity networks, map providers, traffic signal infrastructures, and charging stations all play a role. If the wireless network is unavailable or congested, vehicles can postpone updates, resulting in older code being maintained for longer than desirable. Likewise, if the cloud map service changes its data schema, the vehicle software may need to update in sync to avoid errors. The integration of these systems means that OTA updates sometimes require co-release planning: The back end and the vehicle must evolve together. Operators must maintain compatibility layers and schedule coordinated updates when changes occur in external infrastructure.

Physical infrastructure also matters. AV depots provide charging, cleaning, and maintenance. Updates that require sensor recalibration or hardware alignment cannot be performed OTA; they require bringing vehicles to the depot. Fleet operators must therefore design update campaigns that combine OTA downloads with scheduled maintenance visits. For example, Waymo's fix for the stationary object mis-scoring issue involved both a software patch and an HD map update; the company implemented these in its central garage to ensure that sensors and software remained synchronized.[9] As fleets scale, these operational dependencies become more complex: Limited garage space or charging capacity may dictate how quickly vehicles cycle through updates. Operators can schedule updates across cities or align them with maintenance cycles.

TOWARD RESILIENT OTA FRAMEWORKS: WHAT SHOULD BE BUILT NEXT?

Addressing these challenges requires advances in both technology and process. Several trends point toward a more robust OTA future:

Modular, Software-Defined Architecture

Modern vehicle electronics are shifting from dozens of discrete ECUs to fewer, more powerful domain/zonal controllers that host containerized services behind well-defined interfaces. This enables module-level updates (e.g., perception, mapping, infotainment) with minimal cross-coupling and supports graceful degradation. If a noncritical service fails, the driving core continues while the system reverts or self-repairs. A growing ecosystem is formalizing this software-defined approach: AUTOSAR Adaptive[10] standardizes middleware and service contracts; Scalable Open Architecture for Embedded Edge (SOAFEE)[11] brings cloud native patterns (containers, orchestration) to safety-critical automotive compute; and the open stacks of the Autoware foundation (e.g., Autoware and the OpenAD Kit[12]) provide reference architectures based on ROS 2/DDS for autonomous driving. Complementary initiatives such as Automotive Grade Linux (AGL) and Eclipse SDV/KUKSA[13] further decouple software from hardware and promote interoperable application programming interfaces. Together, these architectures make updates smaller, safer, and faster by isolating services, enforcing versioned contracts, and enabling transactional rollbacks at the module level.

Scalable, Intelligent OTA Delivery Platforms

Cloud-native update services, content delivery networks, and edge caching will enable concurrent updates to thousands of vehicles without overwhelming the networks. Update orchestrators can schedule downloads based on vehicle usage patterns, battery levels, and network conditions, thus minimizing downtime. Peer-to-peer dissemination may handle significant static assets, while critical code remains centrally distributed. Real-time dashboards can track rollout progress, success rates, and anomalies, enabling data-driven decisions about whether to expand or stop deployments.

Simulation and Digital Twins for Validation

The industry is investing in high-fidelity simulation environments and digital twins that mirror real-world cities and vehicles. Before an update is deployed, it can run through these virtual environments to test performance in countless scenarios, including rare combinations of weather, lighting, and traffic conditions.[14] Digital twins can also model interactions with cloud services and traffic infrastructure, revealing dependencies that might otherwise be missed. Combined with on-road A/B testing and shadow mode, simulation forms a continuous validation loop that provides confidence in updates while accelerating iteration.

Continuous Cybersecurity and Incident Response

The OTA pipeline must integrate security monitoring, threat intelligence, and rapid patching. Standards such as ISO/SAE 21434 (road vehicle cybersecurity) and

UNECE R155/R156 codify practices related to risk management, secure updates, and continuous monitoring. Some researchers are exploring distributed ledger technologies to record update provenance, allowing vehicles and regulators to verify that software originates from an authorized source. Although blockchain may not replace public-key infrastructure, it could add an immutable audit trail for multiparty supply chains.

Collaborative Regulation and Certification

Regulators are increasingly recognizing that software updates alter vehicle performance and may require oversight. In some jurisdictions, critical updates must be reported as recalls or even preapproved by authorities. To avoid regulatory bottlenecks, the industry and regulators are exploring model-based certification, where simulation evidence and digital twin results may substitute for some physical testing, provided the models are validated. Close collaboration and transparent data sharing will enable regulators to balance safety assurance with the agility required for OTA updates.

Reinforcement-Learning-Augmented OTA Scheduling

One of our recent studies demonstrates that machine learning can play a role in optimizing OTA delivery itself.[15] In this work, we formulate a new reinforcement learning (RL) framework, as shown in Figure 4, which treats the OTA update process as a Markov decision process and trains an agent to minimize patch size and runtime memory overhead. By treating the software image as a sequence of data blocks and employing a proximal policy optimization (PPO) agent in a custom OpenAI Gym environment, the system learns to select update actions, such as block-level modifications or differential patches, based on network conditions and system constraints. The result is an approximately optimal payload that is smaller and uses less memory than existing methods. This RL-based approach has so far been applied to OTA updates for the entire system (rather than just specific modules), showing promising improvements in bandwidth efficiency and demonstrating how adaptive, learning-based policies could complement traditional compression and scheduling algorithms. Integrating such adaptive policies into OTA platforms could help fleets dynamically choose the most efficient update strategy given current connectivity and hardware conditions, providing another layer of resilience in large-scale deployments.

FUTURE DIRECTIONS FOR OTA RESEARCH

We outline a set of future research directions that aim to improve the reliability, security, and efficiency of automotive OTA updates. Each of these directions targets a key aspect of the OTA ecosystem that can be enhanced as vehicles become increasingly software-defined and connected.

Standardize OTA Protocols and Open Frameworks

Fragmented proprietary OTA stacks across OEMs and suppliers create versioning issues, integration friction, and inadequate security. A standard set of update formats, dependency metadata, and security baselines would enable cross-vendor interoperability and consistent assurance, reducing duplicated engineering so teams can focus on higher-value innovation and deliver faster, safer updates at fleet scale.

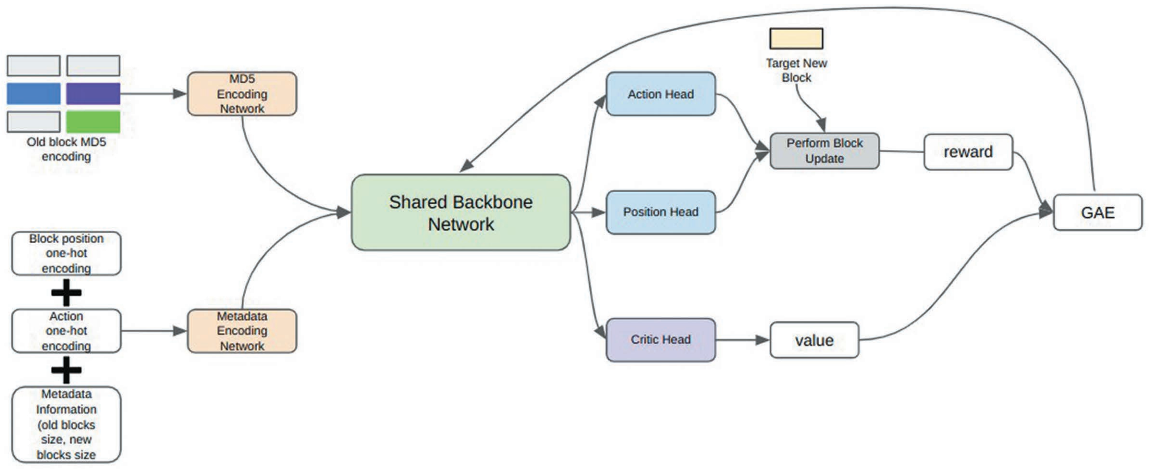

FIGURE 4. Demonstration of the PPO network models used.

Leverage Formal Verification Alongside Simulation

Beyond simulation and road tests, integrating lightweight formal methods (e.g., model checking) into the OTA pipeline can demonstrate that safety-critical properties—such as timing deadlines and interface contracts—remain intact after an update. This adds mathematical assurance where tests may miss corner cases, reducing the risk of subtle regressions in ECUs and complementing staged validation with stronger, automatable guarantees.

Develop Self-Healing Software Architecture

Move beyond coarse A/B rollbacks to autonomous fault detection, isolation, and recovery at the module level—e.g., identifying a hung driver, sandboxing or restarting the component, or reverting only the affected package. Such self-healing runtimes prevent fault cascades in tightly coupled systems, reduce downtime, and keep vehicles safe and operable during and after OTA updates—turning the car into an active agent in its own maintenance.

Optimize Energy and Network Usage

Reduce OTA cost and latency by shipping delta payloads with compression and chunked/resumable transfers, so only the changed bits are transmitted over the air. Pair this with energy-aware scheduling—prefer Wi-Fi and charging windows, throttle to off-peak hours, and prioritize urgent safety patches—to ease 4G/5G load and avoid noticeable battery drain while maintaining high update velocity.

Ensure Privacy-Preserving Telemetry

Telemetry is essential for validating updates and canary rollouts. Still, it must adhere to a privacy-by-design approach: Collect only what validation requires, anonymize and aggregate data on-device, and avoid using raw identifiers (e.g., report error rates or key performance indicators via randomized IDs). Clear user communication (what is collected, why, and options to opt in/out) plus encryption and access controls sustains trust, enabling a reliable feedback loop without exposing individual drivers.

Prepare for Next-Generation V2X Connectivity

Design OTA pipelines to be multichannel by default, allowing for seamless switching among cellular (5G/6G), Wi-Fi offload, roadside edge caches, and vehicle-to-everything (V2X) links to leverage future ultra-low-latency, high-throughput networks. Enable secure cooperative distribution (e.g., car-to-car relays of noncritical assets with attestation and rate limits) so fleets can quickly propagate patches in areas with poor coverage while easing core-cloud load.

CONCLUSION

OTA is no longer optional for autonomous fleets; it is the only practical way to keep complex, safety-critical software up to date at scale. The core challenges are clear: large codebases, tightly coupled components, fleet-wide logistics, uneven connectivity, and persistent cyber risk. However, these challenges are tractable when OTA is treated as an end-to-end system, rather than a delivery afterthought. Practically, this means modular software, simulation-backed and formally validated, with staged rollouts featuring telemetry-driven gates and rapid rollback capabilities. Additionally, it entails energy- and bandwidth-aware distribution, as well as defense-in-depth security. With these foundations, OTA becomes a repeatable safety mechanism rather than a liability, allowing AVs to improve continuously, quickly contain faults, and maintain public trust as the industry transitions to software-defined vehicles.

REFERENCES

1. D. Shepardson and D. M. Sophia, "Waymo recalls 1,200 self-driving vehicles in US after minor collisions," *Reuters*, May 14, 2025. Accessed: Sep. 13, 2025. [Online]. Available: https://www.reuters.com/business/autos-transportation/alphabets-waymo-recalls-over-1200-vehicles-after-collisions-with-roadway-2025-05-14/
2. A. Palmer, "Amazon's Zoox robotaxi unit issues software recall after recent Las Vegas crash," *CNBC*, May 6, 2025. Accessed: Sep. 13, 2025. [Online]. Available: https://www.cnbc.com/2025/05/06/amazon-zoox-recall.html
3. C. Jacobs, "New research shows cars need more memory than a rocket," *Micron*, Dec. 2023. Accessed: Sep. 13, 2025. [Online]. Available: https://www.micron.com/about/blog/applications/automotive/newresearch-shows-cars-need-more-memory-than-a-rocket?
4. C. Souza de Oliveira, R. D. S. Toledo, V. H. Tulux Oliveira Victorio, and A. von Wangenheim, "Trajectory planning for autonomous cars in low-structured and unstructured environments: A systematic review," *IEEE Access*, vol. 13, pp. 48,841–48,871, 2025, doi: 10.1109/ACCESS.2025.3551453.
5. T. S. Ng, *Robotic Vehicles: Systems and Technology*. Singapore: Springer-Verlag, 2021.
6. A. Ghosal, S. Halder, and M. Conti, "Secure over-the-air software update for connected vehicles," *Comput. Netw.*, vol. 218, Dec. 2022, Art. no. 109394, doi: 10.1016/j.comnet.2022.109394.
7. A. Bhattacharjee, H. Mahmood, S. Lu, N. Ammar, A. Ganlath, and W. Shi, "Edge-assisted over-the-air

software updates," in *Proc. IEEE 9th Int. Conf. Collaboration Internet Comput. (CIC)*, 2023, pp. 18–27, doi: 10.1109/CIC58953.2023.00013. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10429931

8. A. Kumar, A. Gholve, and K. Kotalwar, "Automotive security solution using hardware security module (HSM)," SAE Tech. Paper, Tech. Rep., 2024, doi: 10.4271/2024-28-0037. [Online]. Available: https://saemobilus.sae.org/papers/automotive-security-solution-using-hardware-security-module-hsm-2024-28-0037
9. A. M. D. Lee, "Waymo recalls roughly 1,200 self-driving vehicles prone to hitting road barriers," *CBS News*, May 2025. Accessed: Sep. 13, 2025. [Online]. Available: https://www.cbsnews.com/news/waymo-car-recall-software-crash-self-driving/
10. G. Reichart and R. Asmus, "Progress on the AUTOSAR adaptive platform for intelligent vehicles," in *Proc. Automatisiertes Fahren 2020*, Wiesbaden, Germany: Springer-Verlag, 2021, pp. 67–75.
11. "Scalable open architecture for embedded edge (SOAFEE)." SOAFEE. Accessed: Sep. 14, 2025. [Online]. Available: https://www.soafee.io/
12. "OpenAD Kit documentation (version 2.0)." Autoware Universe Documentation. Accessed: Sep. 14, 2025. [Online]. Available: https://autowarefoundation.github.io/open-ad-kit-docs/latest/version-2.0/
13. E. Kuksa, "Kuksa — A digital twin framework for automotive," 2025. Accessed: Sep. 14, 2025. [Online]. Available: https://eclipsekuksa.github.io/kuksa-website/
14. Y. He, H. Chen, and W. Shi, "An advanced framework for ultra-realistic simulation and digital twinning for autonomous vehicles," 2025, *arXiv:2405.01328*.
15. A. Bhattacharjee, Z. Xu, V. Vyas, and W. Shi, "Reles-OTA: A reinforcement-learning enhanced scalable over-the-air update approach for CAVs," in *Proc. IEEE 3rd Int. Conf. Mobility, Oper., Services Technol. (MOST)*, 2025, pp. 241–251, doi: 10.1109/MOST65065.2025.00034.

ARPAN BHATTACHARJEE is a Ph.D. student in computer science in the CAR Lab, University of Delaware, Newark, DE 19713, USA. Contact him at arpan@udel.edu.

WEISONG SHI is an alumni distinguished professor and the chair of the Department of Computer and Information Sciences where he leads the Connected and Autonomous Research Laboratory, University of Delaware, Newark, DE 19713, USA. Contact him at weisong@udel.edu.