The American Journal of Engineering and Technology

ISSN 2689-0984 | Open Access

TYPE Original Research

PAGE NO. 257-261



# Reliability and Recovery Design for OTA Software Updates in Automotive Embedded Systems

**Srikanth Puram**

General Motors Warren Michigan USA

**OPEN ACCESS**

SUBMITTED 19 April 2025
ACCEPTED 19 May 2025
PUBLISHED 30 June 2025
VOLUME Vol.07 Issue 06 2025

CITATION
Konstantin



**Abstract:** Over-the-air (OTA) software updates have become increasingly important in connected and software-intensive vehicles, enabling remote maintenance, security patching, feature updates, and post-deployment system optimization [1], [2], [8]. However, automotive OTA workflows face reliability challenges caused by intermittent connectivity, ignition-cycle interruptions, power-state transitions, limited embedded resources, and cybersecurity governance requirements [2], [6], [7], [8]. This paper presents a reliability-focused design for OTA software update systems in Android-based automotive embedded platforms, emphasizing staged orchestration, checkpoint-based recovery, artifact validation, and controlled failure handling [8], [10], [12], [13]. The proposed architecture incorporates checkpoint-based progress tracking, modular update delivery, cryptographic verification, and deterministic recovery workflows to reduce the risk of incomplete, inconsistent, or unverifiable software update states [3], [5], [8], [9]. The design specifically addresses interruptions such as network disruptions, process restarts, reboot events, and suspend/resume transitions that can disrupt long-running update workflows in embedded automotive environments [2], [8], [13], [14]. The framework provides a practical approach to constructing resilient OTA workflows by combining platform-level update support with application-layer recovery logic, while aligning the update process with expectations in automotive software update engineering, cybersecurity, and software update management [6], [7], [8].

**Keywords:** Over-the-Air (OTA) Updates, Automotive Embedded Systems, Android Automotive, Software Update Reliability, Checkpoint-Based Recovery, Recovery Mechanisms, Connected Vehicles

## 1. Introduction

The automotive industry has increasingly moved toward connected and software-intensive vehicle platforms, where OTA software updates support remote maintenance, feature updates, security patching, and post-deployment optimization [1], [2], [7], [8]. Unlike conventional mobile or enterprise software deployments, automotive update workflows must operate within constrained and safety-sensitive embedded environments, where failed or incomplete updates can result in inconsistent software states, delayed restoration of functionality, or reduced serviceability [1], [2], [8], [9]. Android-based automotive platforms provide useful system primitives for package installation, background task scheduling, update orchestration, and power-state handling, but these platform capabilities must be coordinated with application-level recovery logic for reliable field deployment [10], [12], [13], [14].

Nevertheless, production-grade automotive OTA workflows require additional orchestration beyond platform-provided services, including persistent state tracking, artifact validation, controlled retry behavior, and recovery sequencing after interrupted update attempts [8], [9], [12], [13]. This paper proposes a reliability-oriented OTA design for Android-based automotive embedded systems, emphasizing checkpoint-based recovery, modular update delivery, integrity validation, and deterministic resume logic for interruptions such as network loss, process restart, reboot, and suspend/resume transitions [8], [12], [13],

[14].

## 2. Problem Statement

OTA workflows in connected automotive embedded systems encounter several reliability challenges that must be addressed across the update lifecycle [2], [8], [9]. First, connectivity in connected vehicle environments can be intermittent and variable, which may interrupt package downloads, delay update validation, or require the workflow to resume after network recovery [1], [2], [9]. Second, automotive embedded platforms may transition through ignition cycles, scheduled maintenance windows, low-power states, or suspend/resume conditions, which can interrupt long-running OTA processes and require validated recovery before the workflow continues [8], [13], [14]. Third, automotive embedded systems often operate under storage, memory, power, and background-execution constraints, requiring OTA workflows to validate available resources before download, installation, cleanup, or recovery operations [8], [12], [13]. Fourth, automotive software update systems must satisfy cybersecurity engineering, software update governance, and software update lifecycle expectations, including those reflected in ISO/SAE 21434, UNECE R156, and ISO 24089 [6], [7], [8]. Prior research and standards on automotive FOTA, secure OTA delivery, and road-vehicle software update engineering show that failed updates, rollback risks, incomplete state transitions, and unverifiable artifacts can undermine serviceability, cybersecurity, and user trust in connected vehicles [1], [3], [5], [8], [9].

## 3. Proposed Architecture

The proposed OTA system employs a staged update pipeline consisting of metadata retrieval, package download, artifact validation, installation, activation, and post-update state verification [5], [8], [9], [12]. This staged structure supports failure isolation,

post-interruption state validation, and workflow resumption from the last verified stage, reducing the need to restart the entire update process after connectivity loss, reboot, or execution interruption [2], [8], [9], [13]. Each stage is checkpointed in persistent storage, enabling the OTA workflow to resume from the last verified state rather than repeating the full update sequence after an interruption [8], [9], [13]. This approach aligns with Android's persistent background execution model and session-based installation APIs, while adding application-specific state control needed for production automotive OTA recovery workflows [12], [13].

The architecture supports modular or split-package delivery, enabling targeted updates to specific assets, configuration bundles, application components, or functional modules without requiring the entire update workflow to be repeated [8], [10], [11]. At the platform level, Android's A/B update model and dynamic partition support demonstrate the importance of maintaining a bootable, recoverable, and operationally flexible update state during platform-level software updates [10], [11]. At the application level, the same

reliability principle is extended through smaller update units, explicit validation checkpoints, and deterministic resume logic that verifies the current update state before continuing execution [8], [9], [12], [13].

## 4. Recovery Mechanisms

Recovery is initiated whenever the OTA workflow resumes after network loss, process termination, reboot, or suspend/resume transition, ensuring that the update process continues only after the prior state has been validated [8], [9], [13], [14]. Upon resumption, the system retrieves the persisted checkpoint, revalidates downloaded artifacts against expected metadata, and continues only from stages whose prior state can be verified [5], [8], [9]. This process reduces redundant data transfers and minimizes the risk of partial, stale, or outdated installation states being reused after an interrupted update attempt [5], [8], [9]. In Android Automotive environments, suspend/resume behavior is especially significant because active execution may be paused while update-related state remains partially

preserved, requiring reconciliation between the expected workflow checkpoint and the actual platform state before resuming the OTA process [13], [14].

To strengthen recovery, the design integrates three primary controls: artifact validation, staged rollback or cleanup, and reinitialization of lifecycle-dependent components before the OTA workflow resumes execution [5], [8], [9], [13]. Before resuming installation, the system verifies package hashes, signatures, and expected metadata to confirm that downloaded artifacts match the recorded checkpoint and have not been corrupted, replaced, or partially reused from an invalid state [3], [5], [8], [9]. If temporary files, partially downloaded packages, or intermediate artifacts do not align with the verified checkpoint, the recovery process removes or quarantines them instead of reusing them in a resumed update workflow [5], [8], [9]. The design also reinstates pending work, observers, installation sessions, and status callbacks in a validated sequence, restoring background execution and installation monitoring before the OTA workflow continues [12], [13].

## 5. Security and Integrity Considerations

Reliable automotive OTA systems require integrity, authenticity, rollback protection, and trust controls to ensure that only valid and authorized software artifacts are installed or resumed after interruption [3], [5], [6], [8], [9]. Automotive OTA security literature and software-update standards emphasize authenticity validation, rollback protection, metadata verification, and secure update authorization before an update package is installed or resumed [3], [4], [5], [6], [8], [9]. Signature verification, hash checks, metadata validation, and checkpoint verification are applied before installation or workflow resumption to prevent interrupted, corrupted, outdated, or tampered update artifacts from proceeding in an unsafe state [3], [5], [8], [9]. The system links each update state to trusted metadata and halts resumed operations when the artifact state, package signature, hash value, or checkpoint record does not match the expected update manifest [3], [5], [8], [9]. These checks help prevent unsafe continuation of interrupted, corrupted, outdated, or tampered update workflows, thereby improving the trustworthiness and recoverability of automotive OTA operations [3], [5], [8], [9].

## 6. Implementation Considerations for Android-Based Automotive Platforms

In Android-based automotive systems,

platform-native services are most effective when

combined with application-level coordination that manages update state, validates artifacts, and restores execution context after interruption [8], [12], [13], [14]. WorkManager supports persistent,

constraint-aware background task execution across application restarts and device reboots, while PackageInstaller provides session-based installation control and status callbacks for monitored application deployments [12], [13]. In automotive contexts, these services should be integrated with update-specific constraints, including power-state awareness, network availability, storage prechecks,

installation-session monitoring, and post-resume validation before the update workflow continues [8], [12], [13], [14].

## 7. Evaluation

This paper qualitatively evaluates the proposed design against interruption scenarios commonly encountered in connected automotive OTA deployments, including interrupted downloads, process restarts, deferred installation windows, reboot events, and suspend/resume transitions [2], [8], [9], [14]. For each scenario, the evaluation assessed whether the workflow could resume from a verified checkpoint, whether temporary artifacts could be safely reused or cleaned up, and whether the installation state remained consistent after recovery [8], [9], [12], [13].

Across these scenarios, the checkpoint model reduced unnecessary rework, preserved verified artifacts when safe to reuse, and improved the predictability of resumed installation sequences after interrupted OTA workflows [8], [9], [12], [13]. The design further reduces the likelihood of stale, partial, or unverifiable states persisting after interruption by requiring artifact revalidation, checkpoint verification, and controlled cleanup before the OTA workflow resumes execution [5], [8], [9].

## 8. Discussion

The primary architectural insight is that platform-level OTA support is necessary but not sufficient for reliable delivery of applications,

features, and configurations in connected automotive embedded systems [8], [10], [12], [13]. Production automotive OTA systems require state-aware orchestration that integrates update transport, artifact validation, installation session management, checkpoint recovery, and power state handling [8], [9], [12], [13], [14]. State-aware orchestration is particularly important in connected and

software-intensive vehicles, where update frequency,

software complexity, and post-deployment maintenance needs continue to increase, while failed field updates can affect serviceability, cybersecurity, and user trust [1], [2], [7], [8], [9].

## 4. 9. Conclusion

This paper introduced a reliability-oriented OTA design for Android-based automotive embedded systems, using staged orchestration, checkpoint-based resume logic, modular delivery, artifact validation, and controlled installation recovery [8], [12], [13]. By integrating Android-native installation and background-execution primitives with

application-layer recovery controls, the design improves resilience to network disruptions, process restarts, reboot events, and suspend/resume-related interruptions [12], [13], [14]. Future research should validate this approach through controlled

fault-injection scenarios, recovery-latency measurements, artifact-reuse analysis, and broader evaluation across ECU, application, and

software-package categories [8], [9]. Given the qualitative nature of this evaluation, further work is required to quantify recovery latency, artifact reuse efficiency, checkpoint accuracy, and validation coverage under controlled fault-injection conditions [8], [9].

## References


1. [M. Shavit, A. Gryc, and R. Miucic, “Firmware Update Over The Air (FOTA) for Automotive Industry,” SAE Technical Paper 2007-01-3523, 2007, doi: 10.4271/2007-01-3523.
2. H. Dakroub and R. Cadena, “Analysis of Software Update in Connected Vehicles,” SAE Technical Paper 2014-01-0256, 2014, doi: 10.4271/2014-01-0256.
3. T. K. Kuppusamy, L. A. DeLong, and J. Cappos, “Uptane: Securing Software Updates for Automobiles,” ESCAR USA, 2016.
4. T. K. Kuppusamy, L. A. DeLong, and J. Cappos, “Securing Software Updates for Automotives Using Uptane,” USENIX ;login:, vol. 42, no. 2, 2017.
5. T. K. Kuppusamy, L. A. DeLong, and J. Cappos, “Uptane: Security and Customizability of Software Updates for Vehicles,” IEEE Vehicular Technology Magazine, 2018, doi: 10.1109/MVT.2017.2778751.
6. ISO/SAE 21434:2021, Road vehicles — Cybersecurity engineering, International Organization for Standardization, 2021.
7. UNECE, UN Regulation No. 156 — Software update and software update management system, United Nations Economic Commission for Europe, 2021.
8. ISO 24089:2023, Road vehicles — Software update

engineering, International Organization for Standardization, 2023.

9. ITU-T, Secure Software Updates Over-the-Air for Connected Vehicles, Technical Paper FSTP.SS-OTA, International Telecommunication Union, 2021.

10. Android Open Source Project, “A/B System Updates,” Android Open Source Project documentation, accessed Dec. 2024.

11. Android Open Source Project, “Dynamic Partitions,” Android Open Source Project documentation, accessed Dec. 2024.

12. Android Developers, “PackageInstaller,” Android Developers API Reference, accessed Dec. 2024.

13. Android Developers, “WorkManager,” Android Developers documentation, accessed Dec. 2024.

14. Android Open Source Project, “Android Automotive Power Management,” Android Open Source Project documentation, accessed Dec. 2024.