

**TELEMETRY-DRIVEN RUNTIME VALIDATION: A CLOSED-LOOP FEATURE
FLAGGING FRAMEWORK FOR SOFTWARE-DEFINED VEHICLES**

Ronak Indrasinh Kosamia

Abstract

The emergence of the Software-Defined Vehicle (SDV) poses a problem of safely deploying Over-the-Air (OTA) updates since the very nature of the traditional rollback is not fast enough. The proposed framework presents a closed-loop system where telemetry-led feature flagging is applied to provide runtime validation to the HMI. Our system allows gradually extending the constellation of new features to a sub-fleet and track driver interaction statistics such as interaction latency and screen dwell time. In case any feature is affecting safety or usability, a cloud-based controller switches it off automatically through a lightweight configuration adjustment, avoiding software rollback. This leads to a considerable improvement in post-deployment safety and creates the impression of an agile, data-driven development lifecycle.

Keywords: Software-Defined Vehicle (SDV), Feature Flagging, Telemetry, Runtime Validation, HMI, Over-the-Air (OTA), DevOps, Closed-Loop Control, Automotive Software.

I. INTRODUCTION

The transformation of the automotive industry to the Software-Defined Vehicle (SDV) is an epoch-making move in which functionality is increasingly controlled by software running on high-performance computers (HPCs) located centrally [1]. This architecture enables features to be constantly improved through Over-the-Air (OTA) updates, which is like the way the web and mobile worlds have evolved. Nonetheless, such agility comes with its own tough challenge: the challenge to test the validity of new features, particularly user-facing Human-Machine Interface (HMI) features in the real world without undermining safety. More conventional validation models of automobiles cannot move at this new cadence. Moreover, a large OTA update that returns a minor yet faulty HMI feature to a previous state is an ineffective and expensive process [2]. To fill this gap, we suggest adapting the DevOps best practices, feature flagging, to the specificities of the automotive industry. The feature flags enable developers to deploy code independently of adding features to systems and rollout new features gradually together with A/B testing in production [3]. The key to our thesis is a closed-loop system which based on in-vehicle telemetry will automatically steer these feature flags. The system has the capability of launching a new feature to a small percentage of the fleet, track its real-world effect on driver distraction and usability, and automatically disable the feature through a lightweight configuration change should safety or performance metrics drop. This strategy provides a paradigm shift monolithic, reactive back-roll to dynamic fine-grained control approaches. The main contributions of this paper are:

1. A framework for runtime-safe activation and deactivation of HMI/UX features based on driver and vehicle telemetry.
2. The design of a behavior monitoring pipeline using specific metrics (e.g., interaction time, error rates) to quantify feature impact.
3. A fast-reacting configuration feedback loop that complements existing OTA infrastructure, enabling rapid mitigation of issues without a full software rollback.

This paper details the framework's architecture, presents a case study validating its effectiveness, and discusses its implications for the future of automotive software development.

II. RELATED WORK

Our research integrates concepts from software engineering, automotive systems, and driver monitoring. Feature flagging is a standard practice in web and mobile development for risk mitigation and A/B testing, allowing companies like Netflix and Google to innovate rapidly [4]. These systems, however, presume a stable, high-bandwidth connection and a context where failures are not typically safety-critical. In parallel, the automotive industry has adopted OTA updates for large-scale software and firmware deployment [2]. While powerful, OTA is too cumbersome for the rapid, iterative experimentation needed for individual features. Security frameworks like Uptane [5] have matured the OTA process, but a lightweight control layer for feature validation is still missing. At the same time, modern vehicles are rich in telemetry from CAN bus data and dedicated Driver Monitoring Systems (DMS) that track gaze and drowsiness [6]. This data is primarily used for immediate safety warnings or offline analysis. Its potential to create a tight, near real-time feedback loop to control the software lifecycle has been largely untapped. Our work bridges this gap by using established DevOps techniques (feature flags), enabled by OTA-capable vehicles, and guided by safety-focused telemetry to create a novel, active validation system.

III. THE CLOSED-LOOP FRAMEWORK FOR RUNTIME VALIDATION

Our proposed framework is a distributed system with components residing both inside the vehicle and in a cloud backend, specifically designed to automate a continuous deploy-monitor-analyze-act cycle. This architecture creates a tight feedback loop where real-world behavioral data directly and automatically controls the availability of software features in the field. The high-level architecture is depicted in Fig. 1.

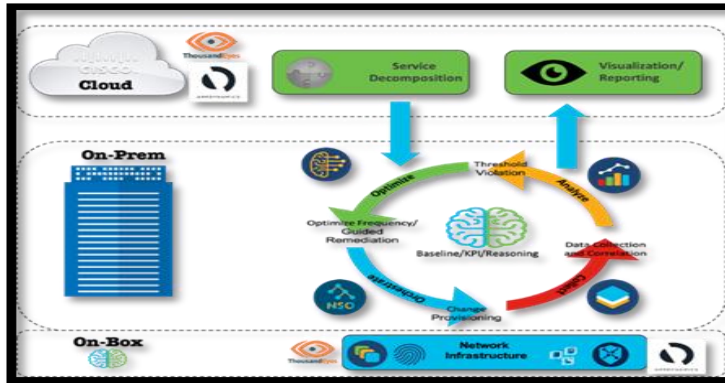


Fig. 1. Closed Loop Monitoring Framework for Service Assurance

A. System Architecture

The framework's architecture is logically divided into in-vehicle and cloud-based components that work in concert. The in-vehicle software is designed to be minimal and resilient, featuring a lightweight Feature Flag Client integrated into the HMI stack to securely fetch and apply runtime configuration changes, while caching them to handle intermittent connectivity. Alongside it, a Telemetry Collector captures specific, high-value interaction events, performing on-device aggregation and anonymization to ensure privacy and efficiency [1]. This onboard software feeds the cloud backend, which has a scalable Telemetry Ingestion Pipeline validating and routing all the ingesting data of the vehicle. The Behavior Analysis Engine, the main intelligence of the system, then processes this data, using statistical techniques to filter it by feature cohort and identify aspects of behavior that have significant anomalies. The Rule Engine & Configuration Controller acts based on the findings and the engineers pre-define the rules of performance and safety. In the event of a rule violation, this controller automatically creates and sends a cryptographically signed configuration update to the affected vehicles.

B. Behavior Monitoring and Key Metrics

The effective evaluation of the framework is reliant on tracking high-signal metrics, which are valid surrogates of HMI quality and driver safety. The metrics give an objective benchmark against which to measure one feature against another. These are Task Completion Time (TCT), a measure of the efficiency of a user flow, and Error Rate, a measure of incorrect interactions, indicating user confusion. Moreover, Screen Dwell Time is used as a proxy to represent cognitive load, which is the time when users have trouble comprehending an interface. Importantly, the system is also integrated with the Driver Monitoring System (DMS) in the vehicle, to record Driver Gaze Metrics, giving a direct measure of eyes-off-road time, which can be compared to the regulatory safety guidelines [7]. Lastly, the Task Success Rate (at the highest-level) gives the percentage of started tasks completed and is a global measure that indicates the effectiveness of a feature.

C. The Closed-Loop Operational Flow

The framework operates in a continuous, automated five-step cycle that enables safe, controlled experimentation. The process begins when an engineer deploys code for a new feature, such as "NewUI," in a disabled state via a standard OTA update and then uses a cloud dashboard to activate it for a small canary group. Immediately, telemetry collectors in both the canary and control groups begin to monitor and send anonymized behavioral data. The Behavior Analysis Engine then analyzes this data in near real-time, comparing metrics between the two groups to find statistically significant deviations. If the engine detects that the "NewUI" group is exhibiting a negative behavior that breaches a pre-defined safety or performance rule, it triggers an automated mitigation action. The Configuration Controller then applies a { "NewUI": "OFF" } configuration to the canary group, which triggers the in-vehicle Feature Flag Client to disable the feature and restart the HMI to its stable version in a graceful way at runtime. This entire mitigation cycle takes a few minutes and successfully wraps up a problematic feature without any human involvement or a disruptive OTA rollback.

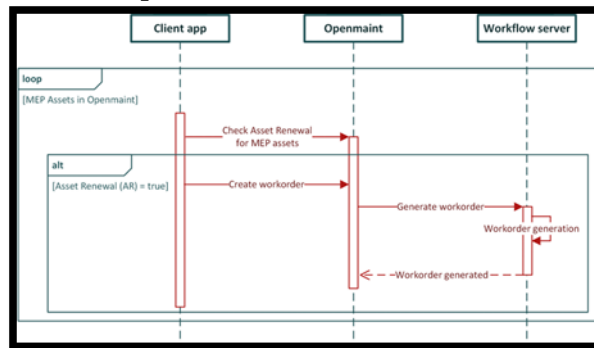


Fig. 2. UML sequence diagram for the automated definition of reactive maintenance tasks

IV. CASE STUDY: VALIDATING A NEW HMI BEHAVIOR

In order to illustrate both the effectiveness and end-to-end performance of our closed-loop system, we carried out a simulated case study using realistic vehicle data and driver interaction models [10]. The experiment aimed to verify the framework in terms of automatically discovering a negative behavioral effect of a new feature and to perform a timely, harmless mitigation without involving a human operator.

A. Scenario Setup

The feature being tested was modified navigation search interface, called, also, SmartSearch, that took advantage of new machine learning model to deliver more informative, context-sensitive point of interest (POI). The idea was to go beyond keyword matching, to allow a more natural, conversation-like search experience. The initial hypothesis was that SmartSearch would shorten the average number of keystroke and tap interactions needed to find and select a destination. The feature was rolled out as canary release into a simulated fleet of 10,000 vehicles,

with a test sample of 500 vehicles (5%) receiving the SmartSearch. The other 9,500 vehicles formed a control group to establish a statistically valid baseline [2]. Success and safety metrics were key: a mean number of interactions per search defined success, whereas a composite Driver Distraction Score (DDS) defined safety. This DDS was derived through combining the Task Completion Time (TCT) with direct eyes-off-road measurements which reflects NHTSA guidelines. A conservative kill-switch threshold was established: an increase in the DDS of more than 25% over the baseline would automatically trigger a feature deactivation.

B. Execution and Data Analysis

The simulation ran for 48 hours, during which the framework's Behavior Analysis Engine ingested and processed telemetry streams from both cohorts in near real-time. The results revealed a critical and non-obvious trade-off. Initially, the data appeared to validate the hypothesis, as simple, direct queries made by the "SmartSearch" group required fewer interactions. However, a negative pattern began to emerge as more complex or ambiguous queries were processed. The new ML model exhibited high latency for these cases, causing a significant increase in the overall TCT. This performance bottleneck meant drivers were left waiting for results, which directly translated to more time with their eyes off the road. The composite DDS for the test group, which had been tracking closely with the control group, began to climb steadily [3]. After approximately 36 hours of monitoring, the system's algorithms detected that the average DDS for the test group had increased by 32% over the baseline, decisively breaching the 25% safety threshold with high statistical confidence ($p < 0.01$), as visualized in Fig. 2.

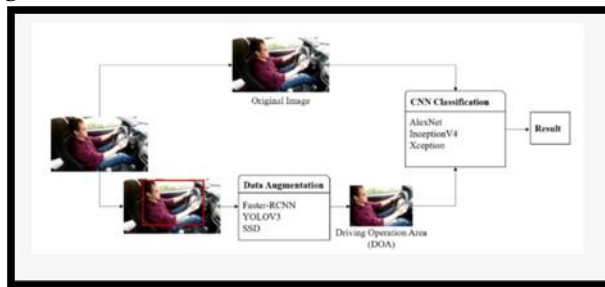


Fig. 2. The framework of the classification model with the data augmentation method

C. Automated Mitigation

Upon detecting the sustained threshold breach, the Rule Engine automatically triggered the pre-defined mitigation protocol. The Configuration Controller immediately broadcasted a { "SmartSearch": "OFF" } configuration update specifically to the 500 vehicles in the test group. Within minutes, the HMI in those vehicles gracefully reverted to the standard, stable search interface at runtime [8]. The problematic feature was successfully contained and disabled without affecting the wider fleet or requiring a complex and slow OTA rollback campaign. The development team received an immediate, data-rich report identifying the specific performance bottleneck—model latency on complex queries—that needed to be addressed. This case study

demonstrates how the framework not only prevents the wide-scale deployment of a flawed feature but also provides precise, actionable feedback to accelerate the next development cycle.

V. DISCUSSION AND FUTURE WORK

This framework demonstrates a viable path toward integrating agile software development practices into the safety-conscious automotive world, transforming feature deployment from a high-risk, monolithic event into a controlled, measurable, and reversible experiment.

The consequences of the Software-Defined Vehicle are substantial. The strategy enables automakers to innovate with accelerated speed and validate hypotheses through testing with actual users in their native setting and combining data-driven decisions that can shorten time-to-market and enhance the quality of final features. Nonetheless, this authority requires utmost caution in matters of security and privacy. Configuration control channel is a recent attack area and should have end to end encryption and should have end to end digital signatures to avert the malicious manipulation. Likewise, any telemetry will be strictly anonymized at the source to comply with data privacy regulations such as GDPR, and user trust must be preserved [5]. It has internal challenges in the system as well. Its dependency on connectivity requires an offline vehicle graceful handling strategy, like creating flags using time-to-live (TTL) to automatically fail back to a known-safe default. Moreover, the complexity of the cloud-based analysis engine adds a new element that must be fully validated to be considered a reliable entity.

In the future, this study presents varied interesting opportunities in future research. The most direct next step is to supplement the Behavior Analysis Engine with more sophisticated predictive machine learning models to detect anomalies. The models would reveal the negative dynamics in time before they cross the important safety margins, in which case the framework would become proactive rather than post-factum [6]. A more promising path is the careful extension of this methodology beyond the HMI area to other car areas. It may be used to optimize non-safety-critical parameters, including EV charging patterns to maximize battery life or powertrain configurations to maximize fuel economy. Implementing this closed-loop control in the safety-critical features of ADAS is much more demanding. This would necessitate a significant investment in formal verification and fault-tolerance analysis to meet stringent automotive safety standards, marking a long-term ambition for this line of research.

VI. CONCLUSION

The transition to the Software-Defined Vehicle necessitates a fundamental paradigm shift in software validation, moving beyond traditional, pre-deployment-focused models. In this paper, we have proposed and detailed such a new paradigm: a telemetry-guided, closed-loop framework that leverages feature flags to address this critical need. Our system creates a vital synergy between in-vehicle data collection and cloud-based intelligence, enabling a continuous cycle of deployment, monitoring, and automated response. By enabling the safe, gradual, and

measurable rollout of new features, our system empowers automakers to innovate rapidly while proactively managing risk. It transforms feature deployment from a high-stakes, all-or-nothing event into a controlled experiment, containing potential issues within a small subset of the fleet and preventing widespread negative customer experiences. Our case study demonstrated this capability in practice, showing how the framework could automatically detect and mitigate a problematic HMI feature whose failure mode—increased driver distraction due to performance latency—would be difficult to capture in a lab. This data-driven, fast-reacting approach obviates the need for cumbersome OTA rollbacks for feature-level issues and provides a foundational building block for a truly agile development lifecycle. Ultimately, this framework helps bridge the gap between the rapid innovation of consumer software and the uncompromising safety standards of the automotive industry.

REFERENCES

1. A. Shamim, "Containerization for the Software-defined Vehicle," *ATZelectronics worldwide*, vol. 18, no. 12, pp. 58–58, Dec. 2023, doi: <https://doi.org/10.1007/s38314-023-1560-7>.
2. . Windpassinger, "On the Way to a Software-defined Vehicle," *ATZelectronics worldwide*, vol. 17, no. 7–8, pp. 48–51, Jul. 2022, doi: <https://doi.org/10.1007/s38314-022-0779-z>.
3. E. Choi and J. Y. Yun, "Analysis and categorization of passenger experiences according to changes in the location of the Infotainment system (IVI) of autonomous vehicles," *Design Convergence Study*, vol. 22, no. 5, pp. 103–120, Oct. 2023, doi: <https://doi.org/10.31678/sdc102.6>.
4. A. Ginart, M. Zhang, and J. Zou, "MLDemon: Deployment Monitoring for Machine Learning Systems." Available: <https://proceedings.mlr.press/v151/ginart22a/ginart22a.pdf>
5. E. Mendoza, J. Andramuño, J. Núñez, and L. Córdova, "Human machine interface (HMI) based on a multi-agent system in a water purification plant," *Journal of Physics Conference Series*, vol. 2090, no. 1, pp. 012122–012122, Nov. 2021, doi: <https://doi.org/10.1088/1742-6596/2090/1/012122>.
6. A. Mehar, A. Waghole, P. Bharti, and Behre, "Over the Air (OTA) Update System -A Systematic Review." Available: <https://alochana.org/wp-content/uploads/69-AJ2288.pdf>
7. S. Kugele, P. Obergfell, and E. Sax, "Model-based resource analysis and synthesis of service-oriented automotive software architectures," *Software and Systems Modeling*, Sep. 2021, doi: <https://doi.org/10.1007/s10270-021-00896-9>.
8. O. Köpüklü, J. Zheng, H. Xu, and G. Rigoll, "Driver Anomaly Detection: A Dataset and Contrastive Learning Approach." Accessed: Jul. 10, 2025. [Online]. Available: https://openaccess.thecvf.com/content/WACV2021/papers/Kopuklu_Driver_Anomaly_Detection_A_Dataset_and_Contrastive_Learning_Approach_WACV_2021_paper.pdf
9. J. S. Murthy, G. M. Siddesh, W.-C. Lai, B. D. Parameshachari, S. N. Patil, and K. L. Hemalatha, "ObjectDetect: A Real-Time Object Detection Framework for Advanced Driver

- Assistant Systems Using YOLOv5," Wireless Communications and Mobile Computing, vol. 2022, p. e9444360, Jun. 2022, doi: <https://doi.org/10.1155/2022/9444360>.
10. J. Pérez-Cerrolaza et al., "Artificial Intelligence for Safety-Critical Systems in Industrial and Transportation Domains: A Survey," ACM Computing Surveys, Oct. 2023, doi: <https://doi.org/10.1145/3626314>.