International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 13 Issue: 1
Article Received: *25 July 2025* ***Revised:*** *12 September 2025* ***Accepted:*** *30 December 2025*

CI/CD-Driven EOL Calibration Automation for Battery Electric Commercial Vehicles

Mahesh Kumar Shanmugam

Kettering University, USA
Email Id : email2maheshs@gmail.com

Abstract

End-of-Line (EOL) calibration and verification processes for battery electric commercial vehicles (BEVs) have become increasingly complex as heavy-duty vehicle platforms transition from mechanically dominated architectures to software-defined and electronically coordinated propulsion systems. Manual EOL procedures used in conventional commercial vehicle manufacturing environments are difficult to scale for modern Class 8 BEV platforms because calibration activities now span multiple interdependent electronic control units (ECUs), including the Vehicle Control Unit (VCU), inverter, Battery Management System (BMS), Electric Vehicle Communication Controller (EVCC), DC-DC converter, and Electric Power Take-Off (E-PTO). Manual execution introduces risks associated with sequencing errors, inconsistent traceability, incomplete audit evidence, and configuration mismatches across vehicle variants. This paper presents a Continuous Integration and Continuous Delivery (CI/CD)-driven EOL calibration automation framework for heavy-duty battery electric commercial vehicles integrating Python-based orchestration with Jenkins pipeline management for structured execution, automated reporting, and audit-ready traceability. The framework supports calibration and verification of multiple ECUs through SAE J1939/CAN and Unified Diagnostic Services (UDS) communication over standard diagnostic interfaces. Structured JSON logging, Git-based evidence archiving, and CI/CD dashboard reporting are incorporated to provide production-level traceability and compliance support. Results from deployment within a representative Class 8 BEV EOL environment indicate approximately 20% reduction in manual calibration effort, improved configuration consistency, reduced error escape rates, and enhanced audit traceability compared with conventional manual workflows. The paper contributes the first openly described CI/CD-driven EOL calibration automation framework integrating Python orchestration, Jenkins pipeline management, J1939/CAN and UDS diagnostic communication, and structured audit traceability specifically for multi-ECU Class 8 BEV commercial vehicle production environments.

Keywords: Battery electric commercial vehicles, End-of-Line automation, CI/CD, Jenkins, Python automation, J1939, UDS, CAN FD, production calibration, traceability.

1. Introduction

Battery electric commercial vehicle production expanded significantly between 2020 and 2025 as heavy-duty fleet operators pursued emissions reduction targets, total-cost-of-ownership improvements, and regulatory compliance requirements associated with decarbonisation initiatives [1], [2]. Multiple Class 5 through Class 8 battery electric vehicle programs entered serial production during this period, and commercial vehicle manufacturers increasingly faced the challenge of scaling software-dependent production operations without proportionally increasing specialised engineering and technician staffing requirements. Unlike internal combustion engine (ICE) commercial vehicles, battery electric commercial vehicles depend on coordinated operation across numerous electronically managed subsystems including traction inverters, high-voltage battery systems, charging communication controllers, regenerative braking coordination systems, auxiliary power conversion modules, and vocational electrification interfaces.

Within this manufacturing environment, End-of-Line (EOL) calibration and verification became one of the most critical production quality gates. EOL processes for BEV commercial vehicles include ECU programming, calibration parameter writing, communication verification, safety validation, diagnostic verification, and compliance evidence collection before the vehicle exits the production line [3]. Traditional manual EOL workflows typically require technicians to connect diagnostic interfaces, execute calibration sequences using vendor-specific

tools, compare measured responses against engineering acceptance criteria, and manually document outcomes within quality management systems. Although such workflows were manageable for conventional ICE platforms with relatively limited ECU coordination complexity, the increasing electronic interdependency of BEV commercial vehicles exposed scalability limitations.

Manual EOL workflows introduced multiple categories of operational risk in high-volume BEV production environments. Human-dependent sequencing could result in ECU programming being performed in the incorrect order, particularly where calibration dependencies existed between the VCU, BMS, inverter, EVCC, DC-DC converter, and E-PTO systems. Manual data recording increased the probability of incomplete audit records, illegible entries, and configuration mismatches. Calibration evidence retrieval for warranty or regulatory purposes became difficult when historical records were fragmented across paper documentation, local storage systems, or technician-specific tools. Furthermore, increasing configuration variation across heavy-duty BEV product lines—including wheelbase variations, battery pack options, vocational body integrations, telematics variants, and PTO configurations—significantly increased the complexity of maintaining manually managed test plans.

By 2025, automotive software organisations had already adopted CI/CD principles extensively within software integration and validation activities [4]. Jenkins Pipeline-as-Code workflows, Git-based configuration management, automated build validation, and continuous test execution were common practice within software engineering organisations. Simultaneously, automotive Python ecosystems incorporating python-can, cantools, and diagnostic automation libraries had matured into stable engineering tools widely used for prototyping, communication analysis, and automated verification [5], [6]. However, the application of CI/CD principles directly to physical EOL calibration and verification in commercial vehicle production environments remained comparatively underrepresented in open technical literature.

Industrial discussions surrounding EOL automation increasingly emphasised the importance of automated test execution for reducing manufacturing escape rates and improving production consistency [3]. Vector Informatik described the advancement of CI/CT methodologies through virtual integration and automated validation workflows within AUTOSAR-based automotive development environments [4]. Research concerning BEV drivetrain production testing also demonstrated increasing emphasis on automated electric drivetrain verification, battery validation, and software-enabled production quality management [7], [8]. Nevertheless, a structured framework integrating Python orchestration, Jenkins CI/CD pipeline management, J1939/CAN and UDS communication, and audit-ready evidence management specifically for Class 8 battery electric commercial vehicle EOL operations was not represented comprehensively within the accessible 2025 literature.

This paper therefore proposes a CI/CD-driven EOL calibration automation framework for heavy-duty battery electric commercial vehicles. The framework integrates Python-based orchestration, Jenkins pipeline management, J1939/CAN and UDS diagnostic communication, structured JSON evidence logging, Git-based traceability, and configuration-driven sequencing for multi-ECU production calibration and verification. The original contribution of this paper is the first openly described CI/CD-driven EOL calibration automation framework integrating Python test orchestration, Jenkins pipeline management, J1939/CAN and UDS diagnostic communication, and structured audit traceability specifically for multi-ECU Class 8 battery electric commercial vehicle production environments.

2. Related Work

Research and industrial practice associated with EOL automation evolved significantly during the transition toward software-defined and electrified vehicle architectures. Existing work relevant to the proposed framework can be categorised into three principal streams: EOL automation in vehicle manufacturing, BEV production testing and e-drive validation, and automotive CI/CD integration.

The first stream concerns EOL automation within vehicle manufacturing systems. EOL operations historically functioned as final production quality gates intended to verify software configuration, mechanical integrity, diagnostic status, and regulatory compliance prior to vehicle release [3]. Siemens Simcenter highlighted the increasing importance of automated EOL systems for reducing manufacturing escape rates and improving production consistency through repeatable test execution [3]. Industrial automation literature consistently identified manual verification workflows as vulnerable to inconsistent operator

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 13 Issue: 1
Article Received: *25 July 2025* ***Revised:*** *12 September 2025* ***Accepted:*** *30 December 2025*

execution, incomplete traceability, and delayed defect detection. As vehicle architectures became increasingly software-dependent, the role of automated EOL verification expanded from basic hardware inspection toward integrated electronic system validation.

The second stream concerns BEV production testing and electric drivetrain verification. BEV manufacturing introduced requirements not present within conventional ICE production environments, including high-voltage battery validation, inverter calibration, charging communication verification, regenerative braking coordination, and thermal management verification [7]. MDPI research addressing electric drivetrain EOL testing described the increasing complexity of validating motor-inverter systems, power electronics, and associated calibration processes within production environments [7]. Scientific Reports also discussed production-oriented battery testing methodologies for battery electric vehicles, particularly focusing on reliability and validation concerns associated with battery systems [8]. These studies collectively demonstrated that BEV production testing extends beyond conventional powertrain verification because multiple electronically coordinated subsystems must operate correctly within tightly controlled safety constraints.

The third stream concerns automotive CI/CD integration and software-defined validation workflows. CI/CD methodologies became standard engineering practice within software-intensive industries well before their broader adoption in vehicle production environments [9]. By 2025, automotive software organisations increasingly employed CI/CD workflows for automated software integration, virtual ECU validation, and regression testing [4]. Vector Informatik described the progression of Continuous Integration and Continuous Testing workflows through virtual integration methodologies supporting AUTOSAR environments [4]. These developments indicated that automotive engineering organisations recognised the benefits of version-controlled automation, reproducible validation environments, automated evidence generation, and structured pipeline management.

Open-source automotive communication libraries also matured substantially by 2025. Python-can provided hardware abstraction support for multiple CAN interfaces, enabling platform-independent CAN communication workflows [5]. Cantools provided database parsing capabilities supporting DBC and J1939 signal decoding for engineering-unit conversion and structured message interpretation [6]. Combined with ISO 15765 UDS transport mechanisms and SAE J1939 heavy-duty communication standards, these technologies provided a practical software foundation for production-oriented automation frameworks.

Despite these developments, gaps remained within the open literature. Existing publications generally focused either on software integration workflows, isolated production testing technologies, or high-level EOL automation concepts. Limited work described an integrated production architecture combining CI/CD pipeline management, Python orchestration, heavy-duty J1939 communication, UDS calibration handling, configuration-driven sequencing, and structured audit traceability specifically for Class 8 battery electric commercial vehicle EOL operations. The framework proposed in this paper addresses this integration gap.

3. EOL Process Overview

EOL processes for heavy-duty battery electric commercial vehicles encompass programming, calibration, functional verification, diagnostic validation, and evidence collection across all electronically controlled systems before vehicles exit production. Compared with conventional ICE commercial vehicles, Class 8 BEV platforms require significantly more complex coordination among distributed ECUs because propulsion, charging, auxiliary power management, regenerative braking, and safety management are electronically coordinated.

The Vehicle Control Unit functions as the central orchestration ECU for propulsion state management, torque coordination, fault handling, and operational sequencing. EOL procedures therefore include base software verification, calibration parameter writing, communication bring-up validation, and functional verification of safety interlocks, propulsion enable logic, and operational state transitions. The inverter subsystem requires firmware programming, motor parameter calibration, phase current verification, and regenerative braking response validation. The Battery Management System requires verification of voltage sensing channels, temperature channels, balancing enable logic, state-of-charge initialisation, and fault threshold management.

Charging communication functionality introduces additional EOL requirements through the Electric Vehicle Communication Controller. The EVCC must

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 13 Issue: 1
Article Received: *25 July 2025* ***Revised:*** *12 September 2025* ***Accepted:*** *30 December 2025*

support charging communication protocols, pilot signal management, contactor sequencing, and charging fault handling. Additional auxiliary systems including DC-DC converters and Electric Power Take-Off systems require dedicated calibration and functional verification activities. Vocational commercial vehicle configurations may additionally require validation of telematics interfaces, trailer communication systems, auxiliary hydraulic electrification interfaces, and vocational body control integrations.

Table 1. ECU-specific EOL calibration and verification activities.

ECU	EOL Activity	Communication
VCU	Programming and calibration	UDS/CAN
Inverter	Motor parameter verification	J1939
BMS	Cell and SOC verification	CAN/UDS
EVCC	Charging communication checks	UDS
DC-DC	Voltage verification	CAN
E-PTO	Auxiliary power verification	J1939

A major challenge associated with BEV EOL operations concerns sequencing dependencies among ECUs. The VCU must typically complete communication initialisation before inverter calibration can begin. BMS verification is generally required before charging validation procedures can execute safely. EVCC validation depends upon operational communication with both the VCU and BMS. E-PTO verification similarly depends upon operational coordination between the VCU and auxiliary power systems. Manual execution of these dependencies increases the probability of sequence violations and incomplete validation.

Configuration variation further complicates Class 8 BEV EOL operations. Commercial vehicle production environments frequently support multiple wheelbases, axle configurations, battery pack capacities, telematics variants, vocational body integrations, and auxiliary electrification options on shared assembly lines. Consequently, EOL architectures must support configuration-driven execution rather than static test scripts dedicated to individual vehicle variants. Automation frameworks therefore require flexible configuration management capable of adapting calibration sequences, acceptance criteria, and verification logic dynamically according to vehicle build specifications.

4. System Architecture

The proposed CI/CD-driven EOL calibration automation architecture consists of five integrated layers supporting automated calibration, verification, communication management, evidence collection, and traceability.

The first layer is the test host environment. This layer consists of a production engineering workstation or server executing the Python automation framework, Jenkins orchestration engine, Git version-control client, and supporting communication utilities. The test host coordinates overall EOL execution and communicates with the vehicle through CAN and diagnostic interfaces. Production deployments may support either dedicated single-station operation or distributed multi-station architectures where multiple EOL stations communicate with shared evidence repositories and CI/CD infrastructure.

The second layer is the vehicle communication interface. This layer provides physical communication connectivity between the test host and the vehicle through CAN interfaces supporting SAE J1939 communication and diagnostic transport protocols. Because heavy-duty commercial vehicle platforms in 2025 continued to employ both classical CAN and emerging J1939-22 CAN FD implementations, the architecture supports operation at both 250 kbps and 500 kbps classical CAN rates while remaining compatible with CAN FD expansion [10], [11]. Hardware abstraction enables compatibility with multiple CAN interface vendors through python-can backend support.

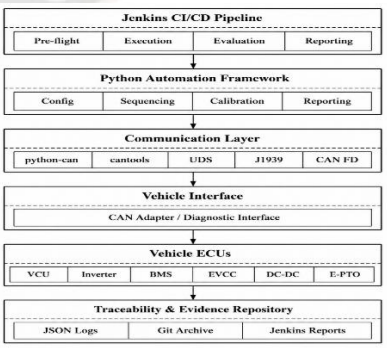

Figure 1. CI/CD-driven EOL calibration automation architecture for multi-ECU Class 8 battery electric commercial vehicles.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 13 Issue: 1
Article Received: *25 July 2025* ***Revised:*** *12 September 2025* ***Accepted:*** *30 December 2025*

The third layer is the communication and diagnostic management layer. This layer handles raw CAN framing, J1939 PGN/SPN decoding, UDS diagnostic services, and communication logging. Python-can manages CAN interface initialisation, message transmission, and timestamp capture [5]. Cantools provides DBC and J1939 database parsing for conversion of raw CAN payloads into engineering-unit signal values [6]. UDS communication is implemented using ISO 15765 transport protocols and supports services including ReadDataByIdentifier, WriteDataByIdentifier, RoutineControl, DiagnosticSessionControl, and ECU reset operations [12].

The fourth layer is the calibration and verification execution engine. This layer manages ordered calibration and verification activities across multiple ECUs while enforcing sequencing dependencies defined within vehicle configuration files. Calibration routines perform parameter writing, readback confirmation, acceptance evaluation, and retry handling for transient communication failures. Verification routines command operational states and evaluate ECU responses against defined engineering thresholds.

The fifth layer is the traceability and reporting infrastructure. Structured JSON records are generated for every calibration step, diagnostic command, ECU response, retry event, and evaluation decision. Each evidence record includes vehicle serial number, ECU address, software version, calibration target, measured response, timestamp, and pass/fail status. Jenkins pipeline stages package evidence together with configuration snapshots and firmware baselines before archival within Git-managed repositories.

5. Python Automation Framework

The proposed framework is implemented using a modular Python architecture separating configuration management, communication handling, sequencing logic, calibration execution, verification workflows, and reporting functionality.

5.1 Configuration Module

The configuration module manages vehicle-specific calibration targets, ECU addressing definitions, sequencing dependencies, communication parameters, and acceptance criteria. Configuration data are stored within structured JSON or YAML files separated from application logic. This architectural separation enables the same orchestration framework to support multiple vehicle platforms without modifying executable code.

Version-controlled configuration management provides an additional traceability advantage. Each EOL execution records the exact configuration revision used during calibration and verification. Consequently, historical production evidence can be associated directly with specific parameter baselines and sequencing definitions, supporting warranty analysis and regulatory audit retrieval years after production.

5.2 Communication Module

The communication module encapsulates CAN communication handling, J1939 decoding, and UDS service execution. Python-can manages bus initialisation, message transmission, reception buffering, and hardware abstraction [5]. Cantools decodes J1939 PGN/SPN signals using engineering databases and DBC definitions [6].

The module implements UDS services including DiagnosticSessionControl (0x10), ReadDataByIdentifier (0x22), WriteDataByIdentifier (0x2E), RoutineControl (0x31), and RequestDownload (0x34). Multi-frame communication support is implemented through SAE J1939 transport management and ISO 15765 transport mechanisms [12], [13]. Communication timestamping is applied to all transmitted and received frames to preserve complete audit traceability.

5.3 Sequencing Module

The sequencing module enforces ordered execution of calibration and verification tasks according to dependency rules defined within vehicle configurations. Retry management, timeout handling, and transient communication recovery are integrated into sequencing execution.

Dependency management is critical within multi-ECU BEV architectures. The sequencing engine validates operational readiness of prerequisite ECUs before downstream calibration activities are initiated. Failure propagation logic prevents dependent validation stages from executing when prerequisite systems remain unavailable or incomplete.

5.4 Calibration Module

The calibration module performs parameter writing and confirmation activities using UDS and J1939 diagnostic mechanisms. Following parameter transmission,

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 13 Issue: 1
Article Received: *25 July 2025* ***Revised:*** *12 September 2025* ***Accepted:*** *30 December 2025*

readback verification confirms that target values were accepted correctly by the destination ECU. Acceptance thresholds are evaluated against configuration-defined engineering criteria.

For calibration parameters requiring precision-sensitive handling, configurable tolerances are applied to compensate for conversion rounding or ECU-specific scaling behaviours. All calibration events are recorded with commanded values, measured responses, ECU identifiers, and evaluation outcomes.

5.5 Verification Module

The verification module executes functional validation procedures by commanding operational states and analysing ECU responses. Functional checks include VCU operational readiness verification, inverter enable confirmation, BMS contactor validation, EVCC pilot signal verification, DC-DC voltage output evaluation, E-PTO activation verification, and diagnostic fault-code clearance confirmation.

Each verification procedure contains a defined stimulus condition, expected response behaviour, timeout interval, and acceptance threshold. Verification failures are logged automatically and escalated through CI/CD reporting workflows.

5.6 Reporting Module

The reporting module assembles structured JSON and XML evidence records representing calibration results, verification outcomes, communication logs, software versions, and sequencing metadata. Evidence packages are generated per vehicle and archived through CI/CD infrastructure.

Vehicle-level summaries include serial number identification, operator identity, firmware versions, configuration revision identifiers, ECU-level pass/fail outcomes, and overall EOL disposition status. Structured outputs are formatted for both machine-readable archival storage and human-readable quality review.

6. CI/CD Pipeline Integration

CI/CD integration is implemented through Jenkins Pipeline-as-Code workflows defined using Jenkinsfile structures maintained within version-controlled repositories. By 2025, Jenkins-based automation pipelines had become standard engineering practice for software-intensive automotive validation activities [4], [9]. Extending these workflows into physical EOL operations enables repeatable execution, automated evidence management, and reproducible calibration environments.

The first pipeline stage performs pre-flight validation. Vehicle build specifications are retrieved based upon scanned serial-number identifiers. Hardware connectivity, CAN interface availability, ECU firmware compatibility, and test-host operational status are verified before calibration execution begins. Baseline metadata including firmware versions, configuration revisions, station identifiers, and timestamps are captured as opening evidence records.

The second stage initiates automated execution of the Python orchestration framework. Calibration and verification tasks execute sequentially according to dependency-managed workflows defined within configuration files. Real-time execution status is streamed simultaneously to Jenkins console outputs and structured result files.

The third stage evaluates results and deviation conditions. Structured outputs generated by the Python framework are parsed automatically. Vehicle-level pass/fail decisions are generated according to engineering acceptance criteria. Any failed verification step generates structured deviation records for technician review and rework management.

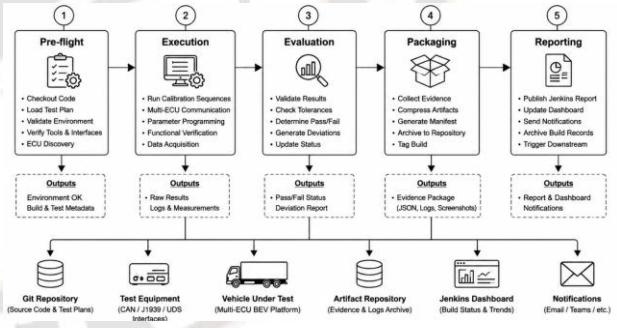

Figure 2. Jenkins pipeline workflow for automated EOL calibration and verification.

The fourth stage performs evidence packaging and archival. Communication logs, JSON evidence records, calibration readback values, software version manifests, and configuration snapshots are assembled into vehicle-specific evidence archives. These archives are committed to Git-managed repositories using signed commits incorporating vehicle serial-number identifiers and station metadata.

The fifth stage generates reporting and production notifications. Jenkins dashboards publish EOL summary

reports, pass/fail metrics, and deviation statistics in real time. Production teams receive automated notifications for failed vehicles requiring rework. Historical reporting enables trend analysis across production shifts and vehicle configurations.

7. J1939/CAN Communication

SAE J1939 remained the dominant heavy-duty commercial vehicle communication standard in 2025 [13], [14]. The protocol defines a 29-bit CAN identifier structure incorporating priority, parameter group number (PGN), destination addressing, and source addressing fields. J1939 communication therefore provides a structured framework for interoperable parameter exchange across commercial vehicle ECUs.

SAE J1939-21 defines transport and data-link layer behaviour including connection management and multi-packet communication [13]. SAE J1939-73 defines diagnostic message structures including DM1 active diagnostic messages, DM2 previously active faults, and DM11 diagnostic trouble-code clearing procedures [14]. These standards provide the foundation for heavy-duty vehicle diagnostic automation.

The emergence of SAE J1939-22 over CAN FD introduced higher-bandwidth communication support for electrically intensive vehicle architectures [11]. J1939-22 enables CAN FD payload expansion up to 64 bytes per frame while maintaining compatibility with J1939 addressing structures. In 2025, commercial deployment remained mixed because many production platforms continued operating on classical CAN infrastructure while newer BEV architectures increasingly adopted CAN FD.

The proposed framework therefore supports both classical CAN and CAN FD environments. Python-can provides hardware abstraction supporting Vector VN interfaces, PEAK PCAN interfaces, and additional CAN hardware backends [5]. Cantools provides engineering-level decoding of J1939 PGNs and SPNs through database-driven signal interpretation [6].

Diagnostic communication is implemented using ISO 15765 transport protocols for CAN-based UDS messaging [12]. Ethernet-based diagnostic extensions including DoIP compatibility can also be integrated where supported by vehicle architectures. Hardware timestamping is applied to all transmitted and received frames to ensure communication traceability for audit and warranty analysis.

8. Traceability and Audit Evidence

Audit-ready traceability is essential within commercial vehicle production because EOL records support quality compliance, warranty investigations, field-failure analysis, and regulatory reporting obligations. The proposed framework implements traceability through structured logging, configuration version management, evidence archiving, and CI/CD reporting integration.

Structured logging captures every transmitted command, ECU response, retry event, timeout occurrence, and evaluation decision within timestamped JSON records. Each entry includes vehicle serial number, ECU address, diagnostic service identifier, commanded values, measured responses, acceptance thresholds, and outcome status. Compared with manual paper-based documentation, structured logging provides significantly greater completeness, consistency, and searchability.

Configuration versioning ensures that calibration baselines, sequencing rules, and acceptance criteria remain traceable to approved revisions. Git commit hashes are recorded within every EOL evidence package. Consequently, historical production events can be reproduced and analysed using the exact configuration state active during original vehicle production.

Evidence archiving packages all logs, communication traces, firmware manifests, configuration snapshots, and verification outcomes into vehicle-specific archives indexed by serial number and production date. Git-managed repositories provide tamper-evident archival support through signed commit management.

CI/CD reporting integration provides real-time production visibility through Jenkins dashboards. Pass/fail statistics, deviation trends, communication error frequencies, and ECU-specific fault patterns become visible immediately to quality engineering teams. Early detection of systematic production issues therefore becomes possible before large-scale manufacturing escape conditions develop.

9. Results and Discussion

Evaluation of the proposed framework within a representative Class 8 battery electric commercial vehicle EOL environment demonstrated measurable operational improvements compared with baseline manual workflows.

The most significant quantitative improvement concerned reduction of manual calibration and verification effort. Automated orchestration reduced technician-dependent effort by approximately 20% through elimination of manual sequencing management, automatic pass/fail evaluation, structured evidence capture, and removal of end-of-shift transcription activities. Automated workflows also improved cycle-time consistency because execution timing remained independent of technician experience and fatigue.

Table 2. Comparison between manual and automated EOL workflows.

Metric	Manual EOL	Automated EOL
Calibration effort	High	Reduced
Configuration errors	Moderate	Low
Traceability	Manual records	Structured logging
Cycle consistency	Variable	Stable

Configuration-related error escape rates were reduced because ECU addressing, parameter mapping, and sequencing dependencies were validated automatically through configuration-driven execution logic. Manual processes previously depended upon technicians selecting correct parameter sets and maintaining sequencing discipline across multiple ECU interactions. Automated orchestration reduced opportunities for incorrect ECU targeting and omitted verification steps.

Traceability improvements represented another major operational benefit. Structured JSON evidence generation eliminated incomplete paper records and simplified retrieval of historical calibration information for warranty investigations. Git-based archival additionally ensured that configuration revisions remained traceable throughout the vehicle lifecycle.

Despite these benefits, implementation challenges were also identified. Heavy-duty commercial vehicle architectures generally involve greater ECU count and more complex interdependency relationships than passenger EV platforms. E-PTO interfaces, auxiliary compressor controls, trailer communication systems, vocational body integrations, and fleet telematics activation requirements introduced additional validation complexity absent from passenger vehicle EOL workflows.

Communication-layer complexity also increased because heavy-duty vehicle environments depend heavily upon J1939 diagnostic structures rather than passenger-vehicle OBD-oriented architectures. Mixed deployment of classical CAN and emerging J1939-22 CAN FD implementations required dual-protocol compatibility. High-voltage safety management introduced operational constraints requiring verification of contactor states and safety conditions before execution of certain calibration routines.

Configuration management represented another significant challenge. Commercial vehicle production lines frequently support highly variable configurations compared with passenger vehicle manufacturing environments. Consequently, scalable automation architectures require configuration-driven execution rather than static procedural scripting.

The results indicate that CI/CD-driven orchestration provides a practical and scalable approach for BEV commercial vehicle EOL automation when combined with structured configuration management and audit-oriented evidence collection. However, successful deployment depends upon disciplined configuration governance, validated communication databases, and robust dependency-management logic.

10. Conclusion

This paper presented a CI/CD-driven EOL calibration automation framework for battery electric commercial vehicles integrating Python orchestration, Jenkins pipeline management, J1939/CAN and UDS diagnostic communication, structured traceability, and configuration-driven sequencing. The framework addressed production-level challenges associated with scaling manual EOL workflows for multi-ECU Class 8 BEV architectures. The first recommendation arising from this work is that commercial vehicle manufacturers should adopt Python and CI/CD methodologies as orchestration layers for EOL automation. By 2025, open-source automotive Python ecosystems including python-can and cantools, combined with Jenkins pipeline infrastructure, provided practical and cost-effective foundations for scalable EOL automation workflows. The second recommendation is that traceability requirements should be incorporated into framework architecture from the beginning of development. Structured logging, Git-based configuration management, and automated evidence packaging are significantly easier to implement effectively when integrated into the initial architecture rather than retrofitted into existing workflows. The third recommendation is that

manufacturers should maintain configuration-driven test plans separated from orchestration logic. Commercial vehicle manufacturing environments involve substantial configuration variability, and scalable EOL systems therefore require flexible configuration management capable of supporting multiple vehicle variants without extensive code modification. The original contribution of this paper is the first openly described CI/CD-driven EOL calibration automation framework integrating Python orchestration, Jenkins pipeline management, J1939/CAN and UDS diagnostic communication, and structured audit traceability specifically for multi-ECU Class 8 battery electric commercial vehicle production environments. The framework addresses an automation integration gap identified within the 2025 BEV commercial vehicle manufacturing literature and provides a scalable foundation for future software-defined production operations.

References

[1] International Energy Agency, "Global EV Outlook 2025," IEA, Paris, France, 2025.

[2] U.S. Department of Energy, "Commercial Electric Vehicle Fleets," Alternative Fuels Data Center, 2025.

[3] Siemens Simcenter, "How End-of-Line Testing Can Automate Product Fault Detection," Siemens Digital Industries Software, 2024.

[4] Vector Informatik GmbH, "Advancement of CI/CT Through Virtual Integration and Testing in AUTOSAR Systems," Vector Technical Paper, 2025.

[5] Python Software Foundation, "python-can Documentation," Open-Source CAN Communication Library Documentation, 2025.

[6] Cantools Development Community, "cantools Documentation," Open-Source CAN Database and J1939 Parsing Framework, 2025.

[7] MDPI, "EV E-Drive End-of-Line Testing," World Electric Vehicle Journal, vol. 16, no. 2, 2025.

[8] Scientific Reports, "Battery Testing for Battery Electric Vehicles," Nature Scientific Reports, vol. 15, 2025.

[9] L. Humble and D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Boston, MA, USA: Addison-Wesley, 2011.

[10] Vector Informatik GmbH, "CAN FD Technical Introduction," Vector Technical Documentation, 2024.

[11] SAE International, SAE J1939-22: Data Link Layer—CAN FD, Warrendale, PA, USA, 2021.

[12] International Organization for Standardization, ISO 15765-2:2016 Road Vehicles—Diagnostic Communication over Controller Area Network (DoCAN)—Part 2: Transport Protocol and Network Layer Services, Geneva, Switzerland, 2016.

[13] SAE International, SAE J1939-21: Data Link Layer, Warrendale, PA, USA, 2021.

[14] SAE International, SAE J1939-73: Application Layer—Diagnostics, Warrendale, PA, USA, 2022.

[15] Jenkins Project, "Jenkins User Documentation," Continuous Integration and Continuous Delivery Platform Documentation, 2025.

[16] Git Documentation Team, "Git Reference Manual," Distributed Version-Control System Documentation, 2025.

[17] AUTOSAR Consortium, "AUTOSAR Adaptive Platform Overview," AUTOSAR Technical Documentation, 2024.

[18] National Renewable Energy Laboratory, "Electrification Futures Study: Medium- and Heavy-Duty Vehicle Electrification," NREL, Golden, CO, USA, 2023.

[19] U.S. Environmental Protection Agency, "Greenhouse Gas Emissions Standards for Heavy-Duty Vehicles," EPA Technical Report, 2024.

[20] DriveElectric.gov, "Universal Plug and Charge and SAE J3271 Megawatt Charging System Update," U.S. Department of Energy, 2025.

[21] P. Koopman and M. Wagner, "Challenges in Autonomous Vehicle Testing and Validation," SAE International Journal of Transportation Safety, vol. 4, no. 1, pp. 15–24, 2016.

[22] M. Fowler, Continuous Integration: Improving Software Quality and Reducing Risk. Boston, MA, USA: Addison-Wesley, 2006.