

SmartOTA-Bench: Replay-Correct and Deployment-Aware Benchmarking of OTA Update Planning

Arpan Bhattacharjee

Dept. of Computer and Information Sciences
University of Delaware
Delaware, USA
arpan@udel.edu

Weisong Shi

Dept. of Computer and Information Sciences
University of Delaware
Delaware, USA
weisong@udel.edu

Abstract—Over-the-air (OTA) update planning for software-defined vehicles and connected autonomous systems is commonly evaluated using transfer size, patch size, or binary-differencing efficiency. These metrics do not establish whether an update plan reconstructs the intended target correctly or can execute within modeled staging, storage, RAM, flash-write, rollback, interruption-recovery, and A/B inactive-slot constraints. This paper presents SmartOTA-Bench, a replay-correct and deployment-aware benchmarking framework that represents each update plan as a structured operation stream, independently verifies byte-for-byte target reconstruction, and simulates execution under explicit device and recovery profiles. Across eight deterministic policies and a bounded five-pair suite comprising one downloaded Alpine release pair, one reproducibly derived rootfs pair, and three deterministic synthetic fixtures, all 1,152 reported configurations are replay-valid. However, only 816 of 864 A/B-enabled configurations are A/B-valid; the remaining 48 exceed the modeled inactive-slot capacity under the capacity-stress streaming profile. Among the evaluated policies, `whole_file_bsdiff` achieves the lowest median transfer, whereas `backup_safe_copy_delta` achieves perfect rollback readiness and ties for the lowest interruption-recovery failure rate. These results demonstrate that transfer efficiency, replay correctness, and modeled deployment feasibility are distinct evaluation properties. SmartOTA-Bench provides an attempt-complete evaluation framework for comparing OTA planners under common correctness, resource, recovery, and deployment contracts; it is not a production OTA stack or vehicle-deployment platform.

Index Terms—Software-defined vehicles, over-the-air (OTA) updates, connected and autonomous vehicles (CAVs), Software Updates, deployment-aware benchmarking, rollback, A/B updates, edge computing

I. INTRODUCTION

Software-defined vehicles (SDVs) and connected and autonomous vehicles (CAVs) increasingly depend on continuous software evolution after deployment. Modern vehicle software stacks include operating system images, embedded Linux root file systems, perception containers, autonomy modules, configuration bundles, and learned-model artifacts. These components must be updated to patch vulnerabilities, correct defects, improve autonomy functions, and adapt deployed fleets to changing operational requirements [1]. Over-the-air (OTA)

update mechanisms make this life cycle feasible by reducing dependence on physical service visits and enabling remote maintenance across distributed vehicle and edge-autonomy platforms [2], [3]. Automotive software-update engineering is also formalized by ISO 24089, which specifies requirements and recommendations for software-update engineering across vehicles, vehicle systems, ECUs, infrastructure, and update-package deployment [4]. At the regulatory level, UNECE Regulation No. 156 establishes requirements for vehicle software-update management systems and the approval of vehicle software updates [5].

OTA update planning is commonly evaluated using transfer size, patch size, or binary-differencing efficiency [6]. These metrics are important because bandwidth remains a practical bottleneck in fleet-scale deployment. However, they are insufficient for SDV and CAV settings, where an update must also satisfy device-level resource and recovery constraints. A byte-small plan may require excessive staging space, exceed transient RAM, write too much flash, overwrite rollback state, fail after interruption, or violate inactive-slot capacity in an A/B update layout [7]. In short, *a byte-small OTA plan can be replay-correct and still be undeployable*.

This paper separates two properties that are often conflated in OTA evaluation. *Replay correctness* asks whether an update operation stream reconstructs the target artifact byte-for-byte from the old artifact. *Deployment feasibility* asks whether the same replay-correct plan can be downloaded, staged, applied, interrupted, resumed, rolled back, activated, and checked within the target device’s bandwidth, storage, RAM, flash-write, checkpoint, rollback, and A/B-slot constraints [8]. Replay correctness is necessary, but it is not sufficient: A correct transformation is not useful if it cannot be safely installed on the target profile.

We present *SmartOTA-Bench*, a replay-correct and deployment-aware benchmark for OTA update planning. SmartOTA-Bench represents each update as a structured operation stream rather than a single patch-size value. A replay validator checks byte-for-byte agreement with the target artifact, while a deployment simulator evaluates the

same stream under explicit profiles that model bandwidth, staging, storage, RAM, flash writes, checkpointing, rollback readiness, interruption recovery, and A/B inactive-slot constraints. An attempt ledger records all benchmark attempts, including completed, incomplete, A/B-invalid, and budget-violating runs. SmartOTA-Bench does not treat the *edge* as an unconstrained cloud server; instead, it models edge deployment by explicitly accounting for bandwidth, staging, storage, RAM, flash-write, checkpoint, rollback, interruption-recovery, and A/B-slot constraints.

In the final deterministic benchmark, all 1,152 reported rows are replay-valid, but deployment semantics expose hidden failures and change policy rankings. Among 864 A/B-enabled rows, only 816 are A/B-valid because replay-valid plans can exceed inactive-slot capacity under a capacity-stress streaming profile. More importantly, the byte-optimal policy, `whole_file_bsdiff`, differs from the safety/rollback-optimal and the best interruption-recovery policy, `backup_safe_copy_delta`. These results show that OTA update planning cannot be evaluated by transfer size alone; replay correctness and deployment feasibility must be measured as separate first-class properties.

This paper makes four contributions:

- We define a replay-correct and deployment-aware OTA benchmark based on structured operation streams, byte-for-byte replay validation, and explicit deployment semantics.
- We introduce a deployment simulator that models bandwidth, staging, storage, RAM, flash writes, checkpoints, rollback readiness, interruption recovery, and A/B inactive-slot constraints.
- We evaluate eight deterministic OTA planning policies on a bounded multi-domain artifact suite, showing that the rankings of transfer-optimal and deployment-safe policies diverge.
- We provide attempt ledger accounting that reports replay valid, deployment complete, deployment-incomplete, A/B invalid, and budget violating runs rather than only successful cases.

SmartOTA-Bench is a benchmark and artifact-evaluation framework, not a production OTA stack. It does not implement secure signing, bootloader-level A/B integration, real-vehicle deployment, cloud rollout orchestration, ECU flashing, or wireless/private 5G delivery. Its purpose is to provide a rigorous evaluation layer for comparing OTA planners under correctness, resource, recovery, and deployment constraints.

II. BACKGROUND AND PROBLEM STATEMENT

SmartOTA-Bench evaluates OTA update planning as a two-stage problem: whether an update plan reconstructs the target artifact correctly, and whether the same plan can be deployed under the resource and recovery constraints of a target device. This section defines the OTA planning model, replay correctness, deployment feasibility, and the reason why transfer size alone is an incomplete evaluation criterion.

A. OTA Update Planning Model

Let O denote the currently installed artifact, and N denote the target artifact. In SDV and edge-autonomy settings, these artifacts may represent operating-system images, embedded Linux root filesystems, container layers, autonomy modules, perception artifacts, or configuration bundles. Given a block size b , an OTA planner receives (O, N, b) and emits a structured operation stream

$$\pi = [op_1, op_2, \dots, op_k], \quad (1)$$

where each op_i may copy existing bytes, insert raw bytes, apply a delta patch, create backup state, verify hashes, checkpoint progress, commit a candidate update, roll back state, or truncate/delete bytes.

This formulation separates the *planner* from the *evaluator*. A planner may be deterministic, heuristic, search-based, or learned, but SmartOTA-Bench requires that it emit a structured operation stream that can be checked by the same replay and deployment machinery.

B. Replay Correctness

Replay correctness is the byte-level correctness contract of an OTA plan. Given O , b , and π , the replay executor reconstructs a candidate target artifact $\hat{N}$. The plan is replay-correct if

$$\text{Replay}_b(O, \pi) = N. \quad (2)$$

SmartOTA-Bench checks this condition byte-for-byte and by comparing the SHA-256 digest of the reconstructed artifact with the target digest.

Replay validation prevents compact but incorrect update streams from being treated as successful. It detects malformed deltas, incorrect source ranges, corrupted raw inserts, unsupported operations, incorrect target lengths, and incorrect truncation or deletion. Therefore, replay correctness is the first gate before any deployment metrics are interpreted.

C. Deployment Feasibility

Replay correctness alone does not determine whether an update can be installed on a target device. We define a deployment profile d as the set of resource, staging, recovery, and activation constraints used during simulation, including bandwidth, staging strategy, RAM budget, persistent storage budget, flash-write model, rollback/backup requirement, checkpoint policy, and A/B inactive-slot capacity.

Given an operation stream π and deployment profile d , the deployment simulator computes deployment metrics, violations, and status:

$$\text{Deploy}(\pi, d) \rightarrow \langle \text{metrics}, \text{violations}, \text{status} \rangle. \quad (3)$$

The metrics include package bytes, network bytes, download time, and installation time; maximum RAM and maximum persistent storage; flash-write bytes; checkpoint behavior; rollback readiness; interruption recovery; and A/B validity. The violations indicate whether the plan exceeds storage,

RAM, flash-write, staging, checkpoint, rollback, interruption-recovery, or inactive-slot constraints. Profile requirements contribute to distinct reported outcomes rather than directly determining terminal simulator completion. In particular, the backup requirement is used to assess rollback readiness and is reported separately from terminal completion.

An OTA plan is deployment-valid for profile d only if it is replay-correct and satisfies all deployment constraints applicable to that profile:

$$\text{Valid}(\pi, O, N, b, d) = [\text{Replay}_b(O, \pi) = N] \wedge \text{DeployOK}(\pi, d). \quad (4)$$

Here, $\text{DeployOK}(\pi, d)$ denotes satisfaction of the applicable staging, storage, RAM, flash-write, checkpoint, rollback, interruption-recovery, and A/B constraints. We distinguish this constraint-level notion of deployment validity from terminal simulation completion. In the evaluation, replay validity, terminal completion, budget compliance, and A/B validity are reported separately because these outcomes are not equivalent: a simulation may reach the completed state while still recording one or more modeled budget violations.

D. Why Replay Correctness Is Not Enough

Many OTA evaluations implicitly assume that a smaller patch is a better patch. This assumption is incomplete for deployment-constrained systems. A `whole_file_bsdifff` policy can emit a compact whole-artifact binary-difference patch and minimize transferred bytes, but the resulting plan may require a large staging space, high transient memory, destructive writes without sufficient backup state, and weak interruption recovery. It can be replay-correct while still being unsafe or infeasible on a constrained target profile. By contrast, a `backup_safe_copy_delta` policy may emit a larger operation stream because it backs up data before destructive writes and then reconstructs the target using copy, delta, or raw-insert operations. This can increase transfer, storage, or flash-write overhead, but it preserves rollback readiness and improves recovery after interruption. The comparison illustrates the core problem that SmartOTA-Bench addresses: the transfer-optimal plan and the deployment-safe plan may differ.

Therefore, OTA update planning should be evaluated along two dimensions: whether the operation stream reconstructs the target artifact exactly, and whether that stream can be staged, applied, interrupted, resumed, rolled back, activated, and checked within the target device budget. SmartOTA-Bench makes both dimensions explicit and measurable.

III. SMARTOTA-BENCH DESIGN

SmartOTA-Bench is designed as a benchmark layer for OTA update planning. Its purpose is not to replace a production OTA stack, but to provide a common evaluation path for planners that emit structured operation streams. Given an artifact pair, a block size, a planning policy, and a deployment profile, SmartOTA-Bench checks whether the resulting update plan correctly reconstructs the target artifact and whether the same

plan can be deployed under explicit resource and recovery constraints.

A. Design Goals

SmartOTA-Bench follows five design goals. **G1: Replay-verifiable correctness** requires each planner to emit an operation stream that can be replayed against the old artifact and checked byte-for-byte against the target artifact. **G2: Deployment-aware feasibility** evaluates replay-valid streams under explicit bandwidth, staging, storage, RAM, flash-write, checkpoint, rollback, interruption-recovery, and A/B-slot constraints. **G3: Policy-agnostic operation interface** separates planning from evaluation so that deterministic heuristics, binary-diff baselines, deployment-aware policies, search-based planners, and learned planners can be compared under the same contracts. **G4: Artifact provenance transparency** records artifact metadata, hashes, sizes, block sizes, domain labels, and source character so that real, derived, and synthetic artifacts are not conflated. **G5: Attempt-complete reporting** records completed, incomplete, replay-invalid, A/B-invalid, timeout, excluded, and budget violating rows to avoid bias. Together, these goals define SmartOTA-Bench as a replay-correct, deployment-aware, policy-agnostic, provenance-explicit, and attempt-complete benchmark for OTA update planning.

B. System Overview

Fig. 1 shows the SmartOTA-Bench evaluation pipeline. The benchmark takes an artifact-pair manifest, old artifact O , target artifact N , block size b , planning policy, and deployment profile as input. The manifest records artifact provenance, size, hash, domain, compression status, and enabled block sizes, while the deployment profile specifies target-device assumptions such as bandwidth, staging, storage, RAM, flash-write behavior, checkpoint policy, rollback requirement, interruption model, and A/B inactive-slot capacity. This manifest-centered representation is consistent with the IETF firmware-update information model, which specifies machine-processable metadata for identifying firmware images, expressing installation instructions, declaring prerequisites, and supporting preinstallation checks [9].

An OTA planner emits a structured operation stream

$$\pi = [op_1, op_2, \dots, op_k], \quad (5)$$

which serves as the common interface between planning and evaluation. SmartOTA-Bench does not require a specific planning algorithm; any deterministic, heuristic, search-based, or learned planner can be evaluated if it emits operations that comply with the benchmark schema. This design ensures that policy comparisons are based on identical replay, deployment, interruption, A/B, and reporting contracts rather than policy-specific evaluation scripts.

C. Structured Operation Schema

SmartOTA-Bench represents each update plan as a structured operation stream rather than an opaque patch-size value.

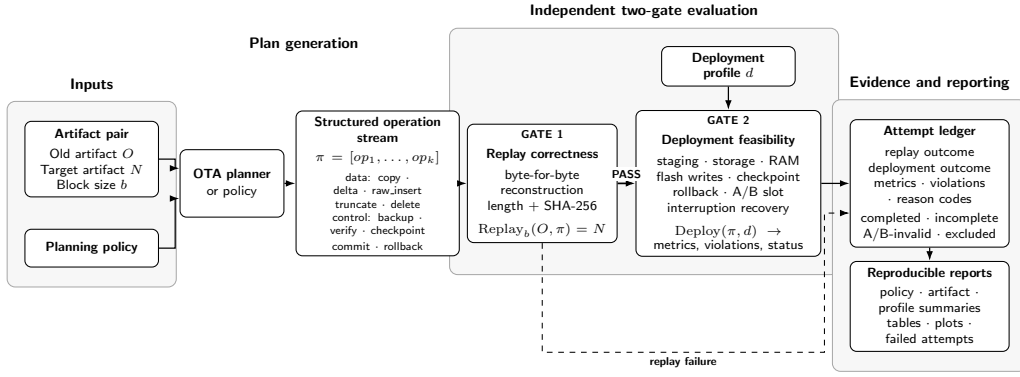


Fig. 1. SmartOTA-Bench evaluation pipeline. Given an artifact pair, block size, and planning policy, the planner emits a structured operation stream. Gate 1 independently verifies byte-for-byte replay correctness and target-digest equality. Only replay-valid plans enter Gate 2, which evaluates the stream under deployment profile d for resource, recovery, interruption, and A/B constraints. Replay failures and deployment outcomes are retained in the attempt ledger for reproducible reporting.

TABLE I
STRUCTURED OPERATION FAMILIES USED BY SMARTOTA-BENCH.

Operation	Purpose
copy	Reuse bytes or blocks from the old artifact or a known source range.
delta	Apply an encoded binary delta between a source region and the target region.
raw_insert	Insert target bytes directly when copy or delta encoding is not useful.
backup	Save old bytes before destructive overwrite to support rollback or recovery.
verify	Check expected hashes or metadata before or after applying operations.
checkpoint	Mark a recoverable intermediate state for interruption recovery.
commit	Finalize the candidate update after successful reconstruction and validation.
rollback	Restore prior state after interruption, failure, or failed activation.
truncate	Shrink the reconstructed artifact to the target length.
delete	Remove obsolete bytes, blocks, or temporary state.

Each operation is a JSON-serializable record with an operation type and operation-specific fields such as source offsets, target offsets, block identifiers, encoded payloads, backup identifiers, or verification hashes. Table I summarizes the operation families. The schema separates data-plane operations that reconstruct target bytes from control-plane operations that support verification, checkpointing, commit, and rollback.

This representation is central to the benchmark: two plans may reconstruct the same target artifact but differ substantially in transfer cost, backup state, checkpoint behavior, rollback readiness, interruption recovery, flash writes, storage pressure, or A/B validity.

D. Replay Validation and Deployment Simulation

SmartOTA-Bench evaluates each operation stream through two layers. The first is the replay validator. Given O , N , b , and π , the validator applies the operations in order and rejects any plan whose reconstructed artifact differs from N byte-for-byte or by SHA-256 digest. This identifies malformed deltas, incorrect source ranges, corrupted raw inserts, unsupported

operations, wrong target lengths, and incorrect truncation or deletion. A compact operation stream is therefore not counted as successful unless it reconstructs the intended target artifact.

The second layer is the deployment simulator. For replay-valid streams, the simulator evaluates installation under a deployment profile d :

$$\text{Deploy}(\pi, d) \rightarrow \langle \text{metrics, violations, status} \rangle. \quad (6)$$

It monitors package bytes, network bytes, download time, install time, peak RAM, peak persistent storage, flash-write bytes, checkpoint behavior, rollback readiness, unsafe overwrites, interruption recovery, A/B-slot validity, and budget violations. For A/B-enabled profiles, the simulator treats the reconstructed target as an inactive-slot candidate and checks slot capacity, candidate validity, activation behavior, boot-health outcome, and rollback-after-failed-boot behavior. This allows SmartOTA-Bench to distinguish replay-invalid plans from replay-valid but deployment-infeasible plans.

E. Attempt Ledger and Reporting

SmartOTA-Bench records every policy-artifact-block-size profile run in an attempt ledger. Each entry stores the attempt status, replay outcome, deployment outcome, A/B outcome, timeout status, exclusion status, and budget-violation metadata. This prevents survivorship bias: failed, incomplete, A/B-invalid, timed-out, or excluded attempts are reported instead of silently removed.

The report-generation pipeline consumes the attempt ledger and benchmark records to produce policy summaries, artifact summaries, deployment-profile summaries, tables, and plots. As a result, the paper reports not only which policies perform well, but also which configurations fail, which constraints are responsible, and how policy rankings change when deployment feasibility is considered.

IV. EXPERIMENTAL METHODOLOGY

This section specifies the experimental matrix used to evaluate SmartOTA-Bench: research questions, artifact pairs,

policies, deployment profiles, block sizes, interruption settings, and metrics. The primary experiment is the final deterministic benchmark over a bounded multi-domain artifact suite. Learned-planner compatibility is a secondary study to support planner-agnostic evaluation and rollout-interface validation, but it is not used to claim learned-method superiority or production OTA readiness.

A. Research Questions

The evaluation is organized around five research questions. RQ1–RQ4 are answered by the primary deterministic benchmark, while RQ5 is secondary.

- **RQ1:** Do evaluated policies emit replay-correct OTA operation streams?
- **RQ2:** Does the transfer-minimizing policy also optimize deployment safety?
- **RQ3:** Which modeled deployment constraints are violated by otherwise replay-correct plans?
- **RQ4:** How do deployment profiles and block sizes affect transfer, flash writes, RAM, storage, and recovery behavior?
- **RQ5:** Can SmartOTA-Bench evaluate learned planners through the same replay and deployment contracts?

RQ1 tests byte-for-byte replay correctness. RQ2 tests the central hypothesis that byte-optimal and deployment-safe OTA planning may select different policies. RQ3 identifies the resource or A/B constraints that make replay-correct plans infeasible. RQ4 studies resource and recovery tradeoffs under different profiles and block sizes. RQ5 checks whether a learned planner can be evaluated through the same structured-operation interface, replay validator, deployment simulator, interruption model, A/B model, and reporting pipeline used for deterministic baselines.

B. Artifact Suite

The deterministic benchmark uses a bounded multi-domain artifact suite covering compressed Linux rootfs artifacts, derived uncompressed rootfs artifacts, container/perception-style artifacts, and autonomy-module-style fixtures. The goal is not to exhaustively represent all production vehicle software, but to evaluate OTA planning behavior across artifact types relevant to software-defined vehicles and edge-autonomy stacks.

Table II summarizes the enabled artifact pairs. The suite separates three evidence levels: P1 is a real downloaded Alpine minrootfs release artifact, P2 is a normalized uncompressed Alpine rootfs tar derived from that artifact family, and P3–P5 are deterministic synthetic fixtures. The synthetic Autoware-style and YOLO/container-style pairs stress OTA mechanics such as changed model regions, copy/delta opportunities, staging, backup, checkpointing, and rollout-interface behavior, but they should not be interpreted as production Autoware artifacts, real YOLO models, proprietary autonomy software, or production vehicle containers.

The 16 KiB and 64 KiB block sizes are evaluated for all enabled pairs. The 4 KiB block size is evaluated only

for the small Autoware-style and YOLO/container-style fixtures. Therefore, 4 KiB results are interpreted as bounded small-fixture sensitivity, not as a full all-domain block-size sweep. The primary deterministic benchmark comprises eight planning policies, 12 eligible artifact-pair/block-size configurations, four deployment profiles, three configured interruption points (25%, 50%, and 75%), and one deterministic seed. The 12 eligible artifact-pair/block-size configurations consist of the 16 KiB and 64 KiB settings for all five artifact pairs, together with the 4 KiB setting for the two small fixtures, P3 and P5. Therefore, the complete experimental matrix contains

$$8 \times [(5 \times 2) + (2 \times 1)] \times 4 \times 3 \times 1 = 1,152$$

unique configurations. The four-field key consisting of policy, artifact pair, block size, and deployment profile has 384 unique values, each appearing exactly three times because the interruption point is varied. Including the interruption point and seed produces 1,152 unique full-configuration keys; no additional repetitions or duplicate full configurations are present.

C. OTA Planning Policies

The evaluated policies are listed in Table III and include whole-file differencing, backup-safe copy/delta planning, deployment-aware greedy planning, copy/delta reuse, copy-only reuse, rolling-hash reuse, blockwise differencing, and full replacement. Each policy receives the old artifact O , target artifact N , and block size b , then emits a structured operation stream evaluated by the same replay and deployment machinery.

Table III summarizes the policy set and records whether each policy is rollback-aware, delta-based, and deployment-aware. A rollback-aware policy explicitly preserves recovery state before destructive writes; a delta-based policy uses binary-difference operations; and a deployment-aware policy uses profile constraints when selecting operations.

These policies are not intended to exhaust the OTA design space. They provide a controlled set for testing whether transfer minimization, rollback readiness, interruption recovery, and deployment feasibility lead to the same or different policy rankings.

a) *Baseline implementation scope:* The `whole_file_bsdiff` and `blockwise_bsdiff` policies use BSDIFF-style binary differencing [10] through the pinned `bsdiff4` version 1.2.6 implementation, retain the generated patch bytes, and reconstruct candidate artifacts using `bsdiff4.patch`. Therefore, their reported transfer volumes are computed from the actual patch payloads generated by this implementation. The `rsync_rolling_hash` identifier denotes a framework-native aligned fixed-block checksum-and-copy policy. Despite its historical name, the policy does not invoke the native `rsync` utility and does not reproduce `rsync`'s sliding-window matching protocol. It computes checksums over aligned blocks and emits copy or insertion operations depending on whether a corresponding reusable block is found.

TABLE II
ENABLED ARTIFACT PAIRS IN THE FINAL DETERMINISTIC BENCHMARK.

ID	Artifact pair	Domain	Source / evidence level	Old → New size	Block sizes
P1	Alpine minirootfs	OS/rootfs	Real release artifact	3.33 → 3.35 MiB	16, 64 KiB
P2	Alpine rootfs tar	OS/rootfs	Derived from real artifact	7.70 → 7.75 MiB	16, 64 KiB
P3	Autoware-style perception module	AV stack	Deterministic synthetic fixture	40 → 50 KiB	4, 16, 64 KiB
P4	YOLO/container multiblock	AV perception container	Deterministic synthetic fixture	3.27 → 3.52 MiB	16, 64 KiB
P5	YOLO/container small	AV perception container	Deterministic synthetic fixture	20 → 20 KiB	4, 16, 64 KiB

TABLE III
DETERMINISTIC OTA PLANNING POLICIES EVALUATED IN THE PRIMARY BENCHMARK.

Policy	Main idea	Rollback?	Delta?	Deploy-aware?
whole_file_bsdiff	Whole-artifact BSDIFF-style patch.	No	Yes	No
backup_safe_copy_delta	Back up old bytes, then reconstruct with copy, delta, and raw insert.	Yes	Yes	Partial
deployment_aware_greedy	Greedy selects operations using deployment and recovery constraints.	Yes	Yes	Yes
copy_delta	Copy identical blocks and delta-encode changed blocks when beneficial.	No	Yes	No
copy_only	Copy identical blocks and raw-insert changed blocks.	No	No	No
rsync_rolling_hash	Framework-native aligned fixed-block checksum-and-copy baseline; does not invoke native rsync or implement its sliding-window protocol.	No	No	No
blockwise_bsdiff	Per-block BSDIFF-style patches where applicable.	No	Yes	No
full_replacement	Send the complete target artifact.	No	No	No

Accordingly, the policy should be interpreted as an rsync-inspired fixed-block reuse baseline rather than as an empirical evaluation of the native rsync tool. Reported runtimes for all policies measure in-process SmartOTA-Bench planning time; they do not represent native-tool execution time, device installation time, or process-level resource consumption.

D. Deployment Profiles

SmartOTA-Bench evaluates every policy under four controlled deployment profiles, summarized in Table IV. The `unconstrained_default` profile provides an unconstrained reference. The three A/B-enabled profiles are `reference_full_staging`, `resource_tight_streaming`, and `capacity_stress_streaming`. These names describe the experimental role of each profile and do not imply correspondence to a particular production ECU.

The profiles are controlled experimental resource envelopes rather than measurements of named ECUs. Their staging and A/B semantics are motivated by established OTA mechanisms, including full-package staging and streaming installation [11], as well as inactive-slot installation, pre-switch integrity verification, post-boot confirmation, and fallback or rollback [12]–[15]. However, their numerical bandwidth, RAM, storage, checkpoint, and slot-capacity values were selected to create progressively restrictive benchmark regimes. Relative to the largest evaluated target artifact, the three A/B-enabled profiles provide storage capacities of approximately 3.10×, 1.55×, and 0.77×, and inactive-slot capacities of approximately 2.06×, 1.03×, and 0.52×, respectively. Consequently, the `capacity_stress_streaming` profile intentionally rejects the 7.75 MiB target when evaluated against its 4 MiB inactive slot. These parameters support controlled sensitivity analysis and should not be interpreted as specifications of production vehicle ECUs.

For reproducibility, the released attempt ledger retains the original internal profile identifiers; this paper uses descriptive display names that emphasize each profile’s experimental role.

For A/B-enabled profiles, the simulator treats the reconstructed target as an inactive-slot candidate and checks whether it fits within the modeled slot capacity. The final deterministic benchmark uses an always-pass boot health setting; therefore, reported A/B failures reflect capacity and activation feasibility rather than forced failed-boot rollback behavior.

E. Metrics

SmartOTA-Bench reports four metric groups. *Correctness metrics* include replay validity, target hash match, and A/B validity. *Efficiency metrics* include package bytes, network bytes, benchmark runtime, and simulated install time. *Resource metrics* include maximum RAM, maximum persistent storage, and flash-write bytes. *Safety and recovery metrics* include rollback readiness, interruption recovery failure, checkpoint resume, and budget violations.

Because artifact sizes and deployment budgets vary across the suite, raw bytes are interpreted together with normalized quantities when useful: **transfer ratio is package bytes divided by target artifact bytes, flash-write ratio is flash-write bytes divided by target artifact bytes, RAM utilization is peak RAM divided by the RAM budget, and storage utilization is peak persistent storage divided by the storage budget.** Together, these metrics allow SmartOTA-Bench to compare OTA policies by correctness, efficiency, resource pressure, and recovery safety rather than by transfer size alone.

Artifact Availability: The SmartOTA-Bench source code, manifests, reproduction scripts, and frozen 1,152-attempt results are available in the versioned public release at <https://github.com/arpanudel/SmartOTA-Bench/releases/tag/smartota-bench-sec2026-artifact-v1.0.0>.

TABLE IV
CONTROLLED DEPLOYMENT-PROFILE ENVELOPES USED IN THE DETERMINISTIC BENCHMARK. VALUES ARE EXPERIMENTAL PARAMETERS RATHER THAN MEASUREMENTS OF NAMED ECUS.

Profile	Bandwidth	Storage	RAM	Staging	Backup	Checkpoint	A/B slot
unconstrained_default	High	High	High	Streaming	No	None	Disabled
reference_full_staging	2,000,000 B/s	25,165,824 B	8,388,608 B	Full package	Required	25 ops	16,777,216 B
resource_tight_streaming	500,000 B/s	12,582,912 B	524,288 B	Streaming	Required	10 ops	8,388,608 B
capacity_stress_streaming	128,000 B/s	6,291,456 B	131,072 B	Streaming	Required	5 ops	4,194,304 B

TABLE V
ATTEMPT-LEDGER ACCOUNTING FOR THE FINAL DETERMINISTIC BENCHMARK. REPLAY VALIDITY, TERMINAL COMPLETION, BUDGET COMPLIANCE AND A/B VALIDITY ARE REPORTED AS SEPARATE, POTENTIALLY OVERLAPPING OUTCOMES. TERMINAL COMPLETION DOES NOT NECESSARILY IMPLY BUDGET COMPLIANCE OR A/B VALIDITY.

Ledger outcome	Count
Attempted configurations	1,152
Reported configurations	1,152 / 1,152
Replay-valid configurations	1,152 / 1,152
Terminal status: completed	1,104 / 1,152
Deployment-incomplete/error	48 / 1,152
Rows with no recorded budget violation	981 / 1,152
Rows with one or more budget violations	171 / 1,152
Timeout	0 / 1,152
Excluded	0 / 1,152
A/B-enabled configurations	864 / 1,152
A/B-valid among A/B-enabled configurations	816 / 864
A/B-invalid among A/B-enabled configurations	48 / 864

V. EXPERIMENTAL ANALYSIS

This section presents the results of the SmartOTA-Bench evaluation. The analysis is organized around the research questions introduced in Section IV-A. The deterministic benchmark is the primary evidence for RQ1–RQ4. The evaluation of the learned method is reported only as secondary evidence for RQ5.

A. RQ1: Replay Correctness and Attempt Accounting

The first question asks whether the evaluated OTA policies emit replay-correct operation streams. Table V summarizes the attempt ledger for the final deterministic benchmark. Across the deterministic benchmark, SmartOTA-Bench attempted and reported 1,152 rows. All 1,152 reported rows were replay-valid, meaning every reported operation stream reconstructed the target artifact byte-for-byte from the old artifact and passed target hash validation. This confirms that the evaluated policies produced correct artifact transformations under the replay validator.

However, replay correctness did not imply terminal deployment completion or satisfaction of every modeled constraint. Of the 1,152 replay-valid configurations, 1,104 reached the terminal status `completed`, whereas 48 ended with a deployment-incomplete or error status. These 48 configurations were not replay failures: their operation streams reconstructed the declared target correctly, but the deployment simulator did not reach its completed state. The profile-level results additionally identify 171 configurations with at least

one recorded budget violation. Because budget violations are tracked separately from terminal status, a completed simulation should not automatically be interpreted as satisfying every modeled resource constraint.

The attempt ledger separates four evaluation dimensions: replay correctness, terminal simulation completion, budget compliance, and A/B validity. These outcomes are reported independently because they are not necessarily mutually exclusive; in particular, terminal completion indicates only that the simulator reached its completed state and does not imply satisfaction of every modeled deployment constraint. The ledger also prevents survivorship bias by preserving completed, incomplete, A/B-invalid, budget-violating, timed-out, and excluded configurations in the benchmark record. All outcome categories should therefore be interpreted using their explicitly reported denominators.

B. RQ2: Transfer-Optimal versus Recovery-Aware Policy Ranking.

RQ2 asks whether the policy that minimizes transfer size also optimizes deployment safety. Table VI summarizes policy-level results across the final deterministic benchmark.

The results show a clear divergence between transfer-optimal and recovery-aware policy rankings. The policy with the smallest median transfer size is `whole_file_bsdiff`, with median transfer 1,097,106 bytes. This confirms that whole artifact binary differencing can be highly effective when the objective is only to minimize transmitted bytes. However, `whole_file_bsdiff` has a rollback-ready rate of 0.250 and an interruption recovery failure rate of 0.625, indicating weak deployment safety under the benchmark’s recovery semantics. By contrast, `backup_safe_copy_delta` is the safety/rollback-optimal policy. It achieves rollback-ready rate 1.000 and the lowest interruption recovery failure rate, 0.042. This policy is not byte-optimal: its median transfer is 1,339,733 bytes, larger than `whole_file_bsdiff`. The additional cost comes from preserving backup state before destructive writes and using a safer operation mix. The result demonstrates the paper’s central finding: minimizing transfer size and maximizing deployment safety require selecting different policies.

Fig. 2 visualizes this ranking divergence. The x-axis reports median transfer size, the y-axis reports the rollback-ready rate, and marker shapes encode the interruption-recovery failure rate. The figure highlights that `whole_file_bsdiff` lies near the transfer-efficient region but has poor rollback and in-

TABLE VI

POLICY-LEVEL METRICS FROM THE FINAL DETERMINISTIC BENCHMARK. ALL POLICIES HAVE REPLAY-VALIDITY RATE 1.000; COMPLETION DENOTES THE FRACTION OF RUNS THAT REACHED THE TERMINAL COMPLETED STATE AND SHOULD NOT BE INTERPRETED AS ROLLBACK READINESS OR FULL BUDGET COMPLIANCE.

Policy	Runs	Completion	Replay	Rollback	Med. transfer B	Budget viol.	Int. fail	A/B valid	Med. flash B	Med. runtime s
whole_file_bsdiff	144	0.958	1.000	0.250	1,097,106	0.333	0.625	0.944	1,780,280	0.869
backup_safe_copy_delta	144	0.958	1.000	1.000	1,339,733	0.146	0.042	0.944	1,954,816	0.429
deployment_aware_greedy	144	0.958	1.000	0.917	1,339,733	0.083	0.042	0.944	1,067,008	0.497
copy_delta	144	0.958	1.000	0.250	2,342,562.5	0.083	0.285	0.944	1,046,528	0.387
copy_only	144	0.958	1.000	0.250	2,464,069	0.083	0.285	0.944	1,046,528	0.022
rsync_rolling_hash	144	0.958	1.000	0.250	2,464,069	0.083	0.285	0.944	1,046,528	0.370
blockwise_bsdiff	144	0.958	1.000	0.250	2,395,960.5	0.083	0.285	0.944	1,046,528	0.361
full_replacement	144	0.958	1.000	0.250	4,154,284	0.292	0.625	0.944	1,780,280	0.028

TABLE VII

GENERATED POLICY WINNERS UNDER EXPLICIT SELECTION RULES. FOR THE “BEST RECOVERY” CRITERION, POLICIES ARE RANKED FIRST BY MINIMUM INTERRUPTION-RECOVERY FAILURE RATE AND, IN THE EVENT OF A TIE, BY MAXIMUM ROLLBACK-READY RATE.

Criterion	Winner	Rollback	Int. fail	Med. transfer B
Safety/rollback	backup_safe_copy_delta	1.000	0.042	1,339,733
Smallest transfer	whole_file_bsdiff	0.250	0.625	1,097,106
Fastest runtime	copy_only	0.250	0.285	2,464,069
Best recovery	backup_safe_copy_delta	1.000	0.042	1,339,733

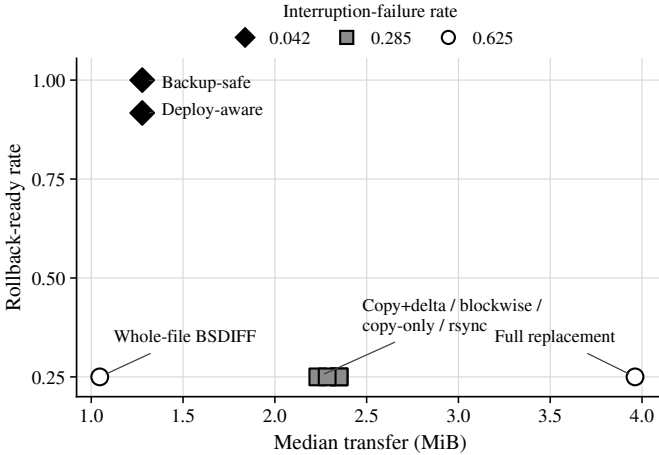


Fig. 2. Median transfer versus rollback readiness across the evaluated deterministic policies. Marker shapes encode the interruption-recovery failure rate: diamond, 0.042; square, 0.285; and circle, 0.625. `whole_file_bsdiff` has the lowest median transfer but low rollback readiness, whereas the backup-safe and deployment-aware policies improve rollback and interruption-recovery behavior. The four copy- and block-reuse policies form a tight cluster; the `copy_only` and `rsync_rolling_hash` points overlap because their plotted values are identical.

interruption behavior, whereas `backup_safe_copy_delta` lies in the deployment-safe region with perfect rollback readiness and low interruption failure.

The policy `deployment_aware_greedy` provides a useful middle ground. It matches the low interruption recovery failure rate of `backup_safe_copy_delta` while reducing median flash writes and budget-violation rate relative to the fully backup-safe policy. In the policy-level summary,

TABLE VIII

A/B VALIDITY BY DEPLOYMENT PROFILE. ONLY A/B-ENABLED PROFILES REPORT A/B-VALID ROWS.

Profile	Rows	A/B rows	A/B valid	Slot-capacity failures
unconstrained_default	288	0	0	0
reference_full_staging	288	288	288	0
resource_tight_streaming	288	288	288	0
capacity_stress_streaming	288	288	240	48

`deployment_aware_greedy` has an interruption failure rate of 0.042, a rollback-ready rate of 0.917, a median flash writes of 1,067,008 bytes, and a budget-violation rate of 0.083. This indicates that deployment-aware policy design can reduce some of the write-amplification and budget pressure introduced by always-backup-safe reconstruction, while still preserving strong recovery behavior.

Overall, RQ2 is answered negatively: the policy that minimizes transfer size does not also optimize deployment safety. SmartOTA-Bench exposes this ranking divergence by evaluating replay-correct operation streams under rollback, interruption, budget, and A/B deployment semantics.

C. RQ3: Deployment Constraint Violations under A/B and Resource Budgets

RQ3 asks which modeled deployment constraints are violated by otherwise replay-correct plans. Table VIII reports A/B validity by deployment profile. A/B metrics are present for 864 rows because the `reference_full_staging`, `resource_tight_streaming`, and `capacity_stress_streaming` profiles enable A/B semantics. Among these A/B-enabled rows, 816 are A/B-valid, and 48 are A/B-invalid. All A/B-invalid rows occur under the `capacity_stress_streaming` profile. These rows are replay-valid, but the target artifact exceeds the modeled inactive-slot capacity. Therefore, the simulator rejects the installation of the inactive slot before activation.

This failure mode should be interpreted carefully. It is expected that an artifact larger than the inactive slot cannot be installed into that slot. The significance is not that this capacity failure is surprising; rather, it shows that SmartOTA-Bench correctly distinguishes byte-level replay validity from deployment-profile feasibility. A replay-only

TABLE IX
BUDGET VIOLATIONS BY DEPLOYMENT PROFILE.

Profile	Rows	Violating rows	Documented pattern
unconstrained_default	288	0	none
reference_full_staging	288	30	storage and RAM pressure
resource_tight_streaming	288	48	storage and RAM pressure
capacity_stress_streaming	288	93	storage, RAM, and A/B slot pressure

benchmark would report these rows as correct updates, while a deployment-aware benchmark exposes that they are infeasible for the selected target profile.

Table IX summarizes budget violations by deployment profile. Budget violations are absent in the unconstrained profile and increase as deployment profiles become more restrictive. The final deterministic benchmark reports 30 violating rows under `reference_full_staging`, 48 under `resource_tight_streaming`, and 93 under `capacity_stress_streaming`. These violations reflect storage, RAM, and A/B slot capacity pressure. This confirms that replay-valid plans can fail because of target-device constraints, even when the byte transformation itself is correct.

D. RQ4: Resource Tradeoffs

RQ4 asks how deployment profiles and block sizes affect transfer, flash writes, RAM, storage, and recovery behavior. Within the evaluated benchmark matrix, recovery-oriented policy choices exhibit measurable resource tradeoffs. Backup-aware policies improve rollback readiness and interruption recovery, but they can increase flash-write volume and storage pressure. Whole-file patching can reduce transfer size, but it can also increase transient-memory or staging requirements and provide weaker rollback behavior.

Fig. 3 compares mean transfer size with mean flash-write volume across the eight deterministic policies. Unlike Table VI, which reports policy-level medians, Fig. 3 uses means over all 144 configurations evaluated for each policy. The figure complements Fig. 2 by exposing the flash-write cost associated with different policy choices.

The `backup_safe_copy_delta` policy achieves a rollback-ready rate of 1.000 but incurs the highest mean flash-write volume, 4,555,016 bytes. In comparison, `deployment_aware_greedy` achieves a rollback-ready rate of 0.917 while reducing mean flash writes to 3,810,104 bytes. The `copy_delta` and `blockwise_bsdiff` policies lie close in the transfer-flash-write space. The `copy_only` and `rsync_rolling_hash` points overlap because both have a mean transfer size of 5,402,484 bytes and a mean flash-write volume of approximately 2,307,091 bytes. These results show that transfer efficiency, flash-write cost, and rollback readiness are distinct evaluation dimensions within the evaluated benchmark matrix.

Additional deterministic diagnostics, including bounded block-size sensitivity and compact operation-mix results, are presented in Subsection V-F. The 4 KiB setting was evaluated only on the small Autoware-style and YOLO/container-style fixtures and therefore provides bounded small-fixture sensi-

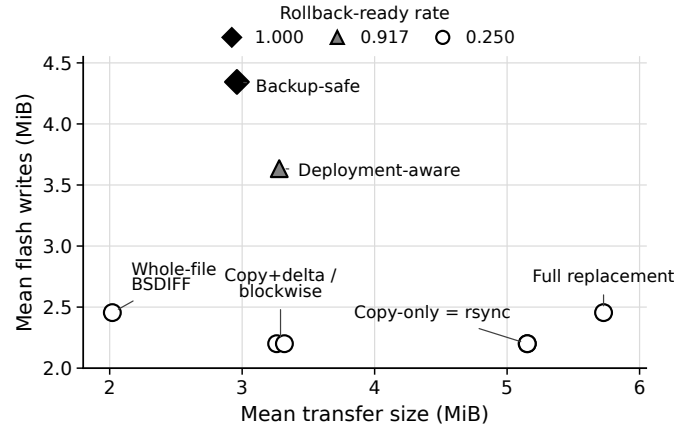


Fig. 3. Mean transfer size versus mean flash-write volume across the eight deterministic policies. Each policy aggregates 144 benchmark configurations. Marker shapes encode rollback-ready rate: diamond, 1.000; triangle, 0.917; and circle, 0.250. Marker size is fixed because every policy has the same deployment-completion rate (0.958). The `copy_only` and `rsync_rolling_hash` points overlap because their plotted means are identical.

tivity evidence rather than a complete all-artifact comparison. The 16 KiB and 64 KiB settings cover all enabled artifact pairs and provide the primary block-size comparison.

E. RQ5: Learned Planner Compatibility

RQ5 asks whether SmartOTA-Bench can evaluate learned planners through the same replay and deployment contracts used for deterministic policies. This study is secondary: its purpose is to test benchmark compatibility, not to claim learned-method superiority.

The bounded learned-method evaluation uses `imitation_bc`, a behavior-cloned operation-family policy that emits SmartOTA-Bench structured operations. Because `imitation_bc` emits the same operation representation as deterministic policies, it can be evaluated by the same replay validator, deployment simulator, A/B model, interruption model, and report generator.

The results show that `imitation_bc` produces replay-valid, A/B-valid, rollback-ready, and interruption successful rows in the bounded evaluation. However, it does not outperform deterministic deployment-aware baselines in terms of transfer bytes or flash writes. Therefore, the correct conclusion is compatibility rather than superiority: SmartOTA-Bench can evaluate learned planners when they emit structured operation streams, but the current learned planner is not claimed to be the best-performing policy. Detailed learned planner results are reported in Subsection V-H. PPO remains smoke-only in the current artifact and is not benchmark comparable.

F. Additional Deterministic Results

This subsection reports supporting deterministic diagnostics for the main benchmark in Section V. These results are not separate experiments; they explain block-size sensitivity and operation-level behavior behind the policy-ranking results.

TABLE X

BOUNDED BLOCK-SIZE SENSITIVITY. THE 4 KiB ROWS COVER ONLY THE SMALL AUTOWARE-STYLE AND YOLO-SMALL FIXTURES.

Block B	Rows	Med. transfer B	Med. flash B	Scope
4,096	192	59,149.5	41,984	bounded small fixtures only
16,384	480	4,652,436	2,041,856	all enabled pairs
65,536	480	4,609,833	2,189,312	all enabled pairs

TABLE XI

COMPACT OPERATION MIX BY DETERMINISTIC POLICY.

Policy	Ops	Copy	Delta	Raw insert	Backup
whole_file_bsdifff	264	0.000	0.545	0.000	0.000
backup_safe_copy_delta	26,904	0.087	0.285	0.163	0.465
deployment_aware_greedy	22,779	0.094	0.320	0.218	0.368
copy_delta	14,400	0.107	0.534	0.359	0.000
copy_only	14,400	0.107	0.000	0.893	0.000
rsync_rolling_hash	14,400	0.107	0.000	0.893	0.000
blockwise_bsdifff	14,400	0.107	0.868	0.025	0.000
full_replacement	264	0.000	0.000	0.545	0.000

1) *Block-Size Sensitivity*: Table X reports bounded block-size sensitivity. The 16 KiB and 64 KiB settings cover all enabled artifact pairs, while the 4 KiB setting covers only the small Autoware-style and YOLO/container fixtures. Therefore, 4 KiB results should be interpreted as small-fixture sensitivity evidence rather than a full all-artifact block-size sweep. The result shows that block granularity changes transfer and flash-write behavior. However, because the 4 KiB setting is evaluated only on small fixtures, block-size sensitivity is treated as supporting evidence rather than as a central claim.

2) *Operation Mix*: Table XI summarizes the operation mix emitted by each deterministic policy. The operation mix helps explain the main policy level tradeoffs. Backup aware policies emit explicit backup operations before destructive writes, which improves rollback readiness and interruption recovery but can increase flash-write and storage cost. Whole file policies emit fewer operations and can reduce transfer size, but they provide weaker fine-grained recovery behavior. Block-oriented policies expose more copy, delta, and raw-insert structure, which helps explain the resource trade-offs observed in the main results.

G. Supplemental User-Space Fault Campaign

We separately exercised the implemented update executor through a bounded local user-space fault campaign. These cases are supplemental and are not included in the 1,152-attempt deterministic benchmark. The campaign crossed seven injected fault classes with two local update fixtures, yielding 14 fault cases. The `package_ota` fixture changed package configuration and threshold metadata while retaining the same YOLOv8n ONNX model. The `model_file_ota` fixture changed ONNX metadata only; it did not modify model weights or graph structure. One fault-free baseline installation for each fixture completed and validated successfully, and these two baseline runs are not included in the 14-case denominator.

All 14 injected cases produced their expected outcomes. The 12 expected-failure cases comprised operation interruption, plan corruption, manifest/hash mismatch, an invalid model file, forced inference-validation failure, and insufficient simulated storage, each applied to both fixtures. Every such case left the active slot unchanged. The two resume-after-interruption cases completed successfully and remained replay-valid. The executor represents its A/B slots as user-space directories and activates a validated candidate through atomic symbolic-link replacement. Consequently, this supplement exercises the implemented fail-closed and resume state transitions, but it neither calibrates the simulator’s resource or timing estimates nor constitutes Jetson, hardware-in-the-loop, bootloader-level A/B, ECU-flashing, or production-OTA validation.

H. Learned Planner Compatibility Details

This subsection reports the secondary learned-method evaluation. The purpose of this evaluation is not to claim that the learned planner outperforms deterministic baselines. Instead, it tests whether a learned planner can be evaluated through the same SmartOTA-Bench interface as deterministic policies: structured operation, replay validation, deployment simulation, interruption recovery, A/B-slot checking, and generated benchmark reports.

1) *Imitation-Based Learned Planner*: The learned policy evaluated in this study is `imitation_bc`, a behavior cloned operation family planner. It is trained from expert operation streams generated by deterministic policies. Expert selection scores candidate deterministic baselines using replay/deployment validity, budget violations, rollback readiness, package size, flash writes, peak storage, peak RAM, total time, and operation count. At inference time, `imitation_bc` extracts features from the OTA environment, applies an action mask, emits structured operation payloads, and passes the output through the same replay and deployment evaluation path used by all other policies. The learned-method evaluation is bounded. It includes five artifact pairs, two block sizes, two deployment profiles, one interruption setting, and three learned-policy seeds. The deterministic comparison policies are `whole_file_bsdifff`, `copy_delta`, `backup_safe_copy_delta`, and `deployment_aware_greedy`. The learned method is evaluated only as compatibility evidence: it demonstrates that a learned planner can produce SmartOTA-Bench compatible structured operation streams.

2) *Aggregate Learned-Method Results*: Table XII summarizes the learned method evaluation. `imitation_bc` produced replay-valid, A/B-valid, rollback-ready, and interruption-successful rows in this bounded study. This confirms that the planner can be evaluated by SmartOTA-Bench under the same correctness and deployment contracts used for deterministic baselines. The result should not be interpreted as learned-method superiority. Although `imitation_bc` achieves compatibility with the benchmark pipeline, it does not outperform deterministic deployment-aware baselines on mean transfer bytes or mean flash-write bytes. In particular, its mean transfer and mean flash write cost

TABLE XII
SECONDARY LEARNED-METHOD SUMMARY. THIS TABLE SUPPORTS BENCHMARK COMPATIBILITY FOR `IMITATION_BC`, NOT LEARNED-METHOD SUPERIORITY.

Policy	Family	Rows	Replay	A/B	Rollback	Int. success	Budget viol.	Mean transfer B	Mean flash B	Mean runtime s
<code>backup_safe_copy_delta</code>	deterministic	20	1.000	1.000	1.000	1.000	0.000	2,205,707	2,223,830.4	0.710
<code>copy_delta</code>	deterministic	20	1.000	1.000	0.000	0.800	0.000	2,590,410.8	1,142,192	0.428
<code>deployment_aware_greedy</code>	deterministic	20	1.000	1.000	1.000	1.000	0.000	2,205,705.7	2,223,830.4	1.236
<code>imitation_bc</code>	learned	60	1.000	1.000	1.000	1.000	0.000	2,282,986.0	2,540,451.2	0.648
<code>whole_file_bsdifff</code>	deterministic	20	1.000	1.000	0.000	0.400	0.300	2,109,922.6	1,458,812.8	1.121

are higher than the strongest deployment-aware deterministic policies in this bounded evaluation. The scientific claim is therefore limited: SmartOTA-Bench can evaluate learned planners when they emit structured operation streams, but this specific learned planner is not the best-performing policy.

3) *Pair-Level Comparison*: Table XIII reports pair-level results for the learned planner and deterministic comparison policies. The table shows that `imitation_bc` remains replay-valid, A/B-valid, rollback-ready, and interruption successful across the bounded evaluation matrix. However, the pair level transfer and flash write values also show that deterministic baselines often remain more efficient.

This pair-level view reinforces the main interpretation. Learned planners should be evaluated by the same replay and deployment machinery as deterministic policies, not by training reward or offline loss alone. A learned planner should only be claimed as superior when it improves deployment-aware metrics against strong deterministic baselines under matched artifact pairs, block sizes, deployment profiles, and interruption settings.

VI. DISCUSSION

The results show that OTA update planning should be evaluated as a deployment-constrained systems problem, not only as a binary-differencing or compression problem. SmartOTA-Bench exposes this distinction by separating replay correctness from deployment feasibility and by reporting policy behavior under explicit device profiles.

A. Why Byte Efficiency Is Not Enough

Byte efficiency remains important for fleet-scale OTA delivery because package and network bytes affect rollout time, infrastructure load, and update cost. However, the deterministic benchmark indicates that byte efficiency is insufficient. As shown in Table VI, `whole_file_bsdifff` achieves the smallest median transfer size, but it has weaker rollback readiness and interruption-recovery behavior than `backup_safe_copy_delta`. Table VII and Fig. 2 make this ranking divergence explicit: the byte-optimal policy and the safety/rollback-optimal policy are different. Fig. 3 further shows that deployment safety has a measurable flash-write cost. Thus, transfer size should remain a metric, but it cannot be the only OTA planning objective.

B. Replay Correctness versus Deployment Feasibility

The central conceptual contribution of SmartOTA-Bench is the separation between replay correctness and deployment feasibility. Replay correctness asks whether the operation stream reconstructs the target artifact exactly from the old artifact. Deployment feasibility asks whether the same stream can be staged, applied, interrupted, resumed, rolled back, activated, and checked within the target device profile.

Table V shows that all reported rows are replay-valid, but deployment semantics still expose failures. The clearest case is the `capacity_stress_streaming` profile: as shown in Table VIII, some replay-valid plans are A/B-invalid because the target artifact does not fit the modeled inactive slot. Table IX similarly shows that budget violations appear only when profiles impose concrete storage, RAM, recovery, and A/B constraints. These results demonstrate that a replay-only benchmark can verify byte correctness, but cannot determine whether an update plan is deployable on a constrained target.

C. Implications for OTA Planners

OTA planners should optimize multiple objectives: transfer size, flash writes, maximum RAM, maximum storage, rollback readiness, interruption recovery, A/B validity, and budget violations. Improving one metric can degrade another. For example, Table VI shows that backup-aware reconstruction improves rollback and interruption behavior, while Fig. 3 shows that safer policies can incur additional flash-write cost. Deployment-aware planning therefore must reason about the target profile, not only the old and new artifact bytes.

This also affects learned OTA planners. A learned planner should not be judged only by reward curves, validation loss, or aggregate patch-size reduction. To be comparable in SmartOTA-Bench, it must emit a concrete structured operation stream that can be replayed, validated, simulated, interrupted, resumed, and checked under the same deployment profiles as deterministic baselines. The secondary learned-method results in Table XII show that `imitation_bc` can be connected to this pipeline, but they support compatibility rather than learned-method superiority. Learned planners should only be claimed as better when they improve deployment-aware metrics against strong deterministic baselines under matched artifact pairs, block sizes, profiles, and interruption settings.

D. Implications for SDV/CAV OTA Evaluation

The distinction between byte efficiency and deployment feasibility is especially important for SDVs and CAVs because

TABLE XIII
SECONDARY LEARNED VERSUS DETERMINISTIC RESULTS BY PAIR. THE TABLE IS BOUNDED TO THE LEARNED-METHOD EVALUATION MATRIX AND SHOULD NOT BE READ AS LEARNED-METHOD SUPERIORITY EVIDENCE.

Pair	Policy	Replay	A/B	Rollback	Int. success	Mean transfer B / flash B
Alpine minirootfs	backup_safe_copy_delta	1	1	1	1	8,229,782.5 / 6,996,528
Alpine minirootfs	copy_delta	1	1	0	1	8,213,936.5 / 3,509,360
Alpine minirootfs	deployment_aware_greedy	1	1	1	1	8,229,776 / 6,996,528
Alpine minirootfs	imitation_bc	1	1	1	1	8,287,997.2 / 6,996,528
Alpine minirootfs	whole_file_bsdiff	1	1	0	0	7,983,059 / 3,509,360
Autoware-style synthetic	backup_safe_copy_delta	1	1	1	1	71,985.5 / 92,160
Autoware-style synthetic	copy_delta	1	1	0	0.5	71,753.5 / 51,200
Autoware-style synthetic	deployment_aware_greedy	1	1	1	1	71,985.5 / 92,160
Autoware-style synthetic	imitation_bc	1	1	1	1	93,199.7 / 92,160
Autoware-style synthetic	whole_file_bsdiff	1	1	0	0.5	70,164 / 51,200
YOLO multiblock synthetic	backup_safe_copy_delta	1	1	1	1	2,691,305.5 / 3,964,928
YOLO multiblock synthetic	copy_delta	1	1	0	1	4,631,134.5 / 2,115,584
YOLO multiblock synthetic	deployment_aware_greedy	1	1	1	1	2,691,305.5 / 3,964,928
YOLO multiblock synthetic	imitation_bc	1	1	1	1	2,982,762.8 / 5,545,984
YOLO multiblock synthetic	whole_file_bsdiff	1	1	0	0	2,461,150 / 3,696,640
YOLO small synthetic	backup_safe_copy_delta	1	1	1	1	24,154.5 / 36,864
YOLO small synthetic	copy_delta	1	1	0	0.5	24,038.5 / 18,432
YOLO small synthetic	deployment_aware_greedy	1	1	1	1	24,154.5 / 36,864
YOLO small synthetic	imitation_bc	1	1	1	1	30,581 / 38,912
YOLO small synthetic	whole_file_bsdiff	1	1	0	0.5	24,027 / 20,480
Synthetic grow	backup_safe_copy_delta	1	1	1	1	11,307 / 28,672
Synthetic grow	copy_delta	1	1	0	1	11,191 / 16,384
Synthetic grow	deployment_aware_greedy	1	1	1	1	11,307 / 28,672
Synthetic grow	imitation_bc	1	1	1	1	20,389.3 / 28,672
Synthetic grow	whole_file_bsdiff	1	1	0	1	11,213 / 16,384

failed updates can affect vehicle availability, rollback paths, and field recovery. Vehicles and edge-autonomy platforms may experience interrupted downloads, partial installs, power loss, failed activation, limited storage, constrained RAM, flash-wear concerns, and intermittent connectivity. SmartOTA-Bench captures these risks through rollback-readiness, interruption-recovery, A/B-validity and budget-violation metrics, as reflected in Tables VI, VIII, and IX.

A/B validity is particularly important because a replay-valid update is not deployable if the candidate artifact cannot fit in the inactive slot or preserve a rollback path. More broadly, the bounded block-size results in Table X and the resource tradeoff in Fig. 3 show that policy behavior depends on artifact granularity, device profile, and the deployment metric being optimized. SmartOTA-Bench therefore provides an evaluation layer for OTA planners that reflects the resource and recovery constraints encountered in SDV/CAV deployment settings.

VII. RELATED WORK

SmartOTA-Bench relates to prior work in secure automotive OTA systems, binary differencing and synchronization, rollback-oriented deployment frameworks, learning-based OTA optimization, and systems benchmarking.

A. Automotive OTA, Security, and Deployment Frameworks

OTA update systems are essential for maintaining modern mobile, embedded, IoT, and vehicle software after deployment. In automotive and software-defined-vehicle settings, OTA systems must address not only delivery efficiency but also authenticity, integrity, staged rollout, rollback, and fleet-management requirements [16]. Security frameworks such as TUF and Uptane protect software update delivery against compromised repositories, signing-key compromise, rollback attacks, and other supply-chain threats [17]–[19]. Uptane extends secure updates concepts to ground vehicles by defining automotive software-update roles, metadata, and repository responsibilities for OEM and supplier ecosystems [20]. Production update frameworks further show that deployment safety depends on device layout and recovery mechanisms, not only on patch format. Android A/B and Virtual A/B updates reduce the risk of leaving a device unbootable after an OTA update [21]. Embedded Linux update frameworks such as Mender, RAUC, SWUpdate, Rsync and OSTree/rpm-ostree support dual-rootfs updates, redundant slots, atomic deployment, rollback, and signed artifacts, streaming installation, or previous-version recovery [22]–[27]. SmartOTA-Bench is complementary to these systems: it does not replace secure signing, bootloader integration, or fleet orchestration. Instead, it evaluates whether an OTA planner emits a structured operation stream that

reconstructs the target artifact and remains feasible under resource, rollback, interruption, and A/B-slot constraints. The IETF SUIT architecture similarly defines roles and security requirements for reliable firmware updating on resource constrained devices, including manifest based update descriptions and protected firmware delivery [28].

B. Binary Differencing and OTA Planning

Binary differencing and synchronization techniques reduce transfer cost when two artifact versions are similar. BSDiff-style methods generate compact binary patches [29]; Xdelta provides binary delta compression using VCDIFF-style streams [30], [31]; and native rsync uses weak rolling and strong checksums with a sliding search procedure to identify reusable regions [32]. SmartOTA-Bench does not reproduce native rsync; its `rsync_rolling_hash` policy compares aligned fixed-size blocks. These techniques motivate several SmartOTA-Bench baselines, including whole-file differencing, blockwise differencing, copy/delta reconstruction, and rolling-hash copy-style policies.

However, binary-diff methods primarily optimize byte-level transfer or synchronization efficiency. They do not by themselves determine whether the resulting update plan preserves rollback state, fits within RAM or storage budgets, resumes after interruption, or satisfies A/B inactive-slot constraints. SmartOTA-Bench therefore, treats binary differencing as one family of OTA planning policies, not as the complete evaluation problem. The benchmark evaluates binary-diff, copy-based, backup-aware, and deployment-aware policies under the same replay and deployment contracts.

C. Learning-Based OTA Optimization and Benchmarking

Prior OTA optimization work has explored scheduling, compression, delta-update generation, resource-aware planning, and fleet-oriented delivery. In the context of CAV software updates, ReLES-OTA formulated OTA planning as a reinforcement-learning problem, modeling full-system updates as a sequential decision process and optimizing update payload size and runtime memory overhead using an MDP/PPO-based framework [6]. This shows that OTA planning can be treated as more than a static binary-diff operation: planners can learn or search over update actions under resource-related objectives.

SmartOTA-Bench is complementary to ReLES-OTA. ReLES-OTA focuses on learning an OTA update policy that reduces patch size and memory overhead; SmartOTA-Bench focuses on how deterministic and learned planners should be evaluated once they emit concrete update plans. In particular, SmartOTA-Bench requires planners to emit structured operation streams that can be replay-validated and deployment-simulated under matched profiles. This distinction matters because reward curves or training losses do not prove byte-for-byte correctness, rollback readiness, or interruption recovery, or A/B validity. More broadly, rigorous systems benchmarking requires careful experimental design and reporting to preserve reproducibility and comparability [33]. SmartOTA-Bench follows this principle through artifact manifests, structured oper-

ation streams, replay validation, deployment simulation, and attempt-ledger accounting.

VIII. LIMITATIONS

SmartOTA-Bench has six limitations. First, the artifact suite is bounded: Alpine minrootfs is the only real downloaded release artifact family, the Alpine rootfs tar is derived from that family, and the YOLO/container and Autoware-style artifacts are deterministic synthetic fixtures. These fixtures stress OTA behavior, but should not be interpreted as production YOLO models, Autoware artifacts, or proprietary autonomy software.

Second, SmartOTA-Bench is a deployment-aware simulator rather than a production OTA stack. It models replay correctness, staging, storage, RAM, flash writes, checkpointing, rollback readiness, interruption recovery, and A/B slot feasibility, but does not implement bootloader-level integration, ECU flashing, production signing, cloud rollout orchestration, wireless/private-5G delivery, secure Uptane metadata, or flash-endurance validation.

Third, the A/B model is simplified. The final deterministic benchmark uses an always-pass boot-health setting, so reported A/B failures primarily reflect inactive-slot capacity and activation feasibility rather than forced failed-boot rollback behavior.

Fourth, learned-planner evaluation is limited. The `imitation_bc` experiment demonstrates compatibility with the structured operation interface, replay validator, deployment simulator, interruption model, and A/B model, but it is not evidence of learned-method superiority. PPO is not benchmark-comparable in the current evaluation; GRPO [34], Decision Transformer, GNN-based planners, constrained RL, and hybrid search-plus-learning methods remain future work.

Fifth, the block-size sweep is bounded. The 16 KiB and 64 KiB settings cover all enabled artifact pairs, while 4 KiB is evaluated only on the small Autoware-style and YOLO/container fixtures. Thus, 4 KiB results should be interpreted as small-fixture sensitivity evidence rather than a full all-artifact sweep.

Sixth, ReLES-OTA was not reproduced under a matched SmartOTA-Bench operation-stream interface. Such a comparison would require retraining and interface adaptation. Accordingly, this paper makes no empirical comparison with, or superiority claim over, ReLES-OTA.

IX. CONCLUSION

This paper presented SmartOTA-Bench, a replay-correct and deployment-aware benchmark for OTA update planning. SmartOTA-Bench extends OTA evaluation beyond byte efficiency by representing update plans as structured operation streams, validating byte-for-byte replay correctness, and simulating deployment behavior under explicit resource, rollback, interruption-recovery, and A/B-slot constraints.

The deterministic benchmark shows that replay correctness and deployment feasibility are separate properties. An update plan may reconstruct the target artifact exactly while still

failing deployment constraints on a target profile. The results also show that policy rankings change when deployment safety is considered: the transfer-optimal policy is not necessarily the rollback-safe or interruption-recoverable policy. This supports the central argument of the paper: OTA planners should be evaluated under correctness, resource, recovery, and deployment semantics, not by patch size alone. A separate 14-case local user-space fault supplement also exercised the implemented fail-closed and resume transitions, but it does not provide real-device or hardware-in-the-loop evidence.

Future work will expand SmartOTA-Bench through a larger real-artifact suite covering operating system, container, autonomy-module, and embedded-software updates; hardware in-the loop validation on real edge devices and vehicle relevant targets; production-grade signing, authentication, and bootloader-level A/B modeling; and structured learned planners such as benchmark-compatible PPO, GRPO, constrained reinforcement learning, and hybrid search-plus-learning methods evaluated under the same replay and deployment contracts.

X. ACKNOWLEDGMENT

This work is funded by the U.S. National Science Foundation (NSF) under Grant No. 2426482.

REFERENCES

- [1] S. Aust, "Vehicle update management in software defined vehicles," in *2022 IEEE 47th Conference on Local Computer Networks (LCN)*, IEEE, 2022, pp. 261–263.
- [2] A. Ghosal, S. Halder, and M. Conti, "Secure over-the-air software update for connected vehicles," *Computer Networks*, vol. 218, p. 109394, 2022.
- [3] Y. Luo and S. Lu, "Incremental over-the-air update for software-defined vehicles with c-v2x," in *2025 IEEE 101st Vehicular Technology Conference (VTC2025-Spring)*, 2025, pp. 1–7.
- [4] International Organization for Standardization, "Road Vehicles—Software Update Engineering," International Organization for Standardization, Tech. Rep. ISO 24089:2023, 2023.
- [5] United Nations Economic Commission for Europe, "UN Regulation No. 156: Uniform Provisions Concerning the Approval of Vehicles with Regards to Software Update and Software Updates Management System," <https://unece.org/transport/documents/2021/03/standards/un-regulation-no-156-software-update-and-software-update>, 2021, accessed: Sep. 4, 2026.
- [6] A. Bhattacharjee, Z. Xu, V. Vyas, and W. Shi, "Reles-ota: A reinforcement-learning enhanced scalable over-the-air update approach for cavs," in *2025 IEEE 3rd International Conference on Mobility, Operations, Services and Technologies (MOST)*, 2025, pp. 241–251.
- [7] F. Mahfoudhi, A. K. Sultanian, and J. Famaey, "Over-the-air firmware updates for constrained nb-iot devices," *Sensors*, vol. 22, no. 19, p. 7572, 2022.
- [8] W. Wei, J. Banerjee, S. Islam, C. Pan, and M. Xie, "Energy-aware incremental ota update for flash-based batteryless iot devices," in *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2024, pp. 51–56.
- [9] B. Moran, H. Tschofenig, and H. Birkholz, "A manifest information model for firmware updates in internet of things (iot) devices," RFC Editor, RFC 9124, Jan. 2022. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc9124>
- [10] C. Percival, "Naïve Differences of Executable Code," <https://www.daemonology.net/papers/bsdiff.pdf>, 2003, accessed: Sep. 3, 2026.
- [11] SWUpdate Project, "SWUpdate Best Practice," <https://sbabic.github.io/swupdate/swupdate-best-practise.html>, accessed: Sep. 3, 2026.
- [12] N. Rathor and S. Sinha, "Mender-fpga: An open source framework for fpga remote update for ml applications," in *2024 IEEE International Symposium on Smart Electronic Systems (iSES)*, IEEE, 2024, pp. 30–35.
- [13] Northern.tech, "Introduction to Mender," <https://docs.mender.io/overview/introduction>, accessed: Sep. 3, 2026.
- [14] RAUC Project, "Integration," <https://rauc.readthedocs.io/en/latest/integration.html>, accessed: Sep. 3, 2026.
- [15] Android Open Source Project, "A/B (Seamless) System Updates," <https://source.android.com/docs/core/ota/ab>, accessed: Sep. 3, 2026.
- [16] A. Shoker, F. Alves, and P. Esteves-Verissimo, "Scalable secure over-the-air software updates for vehicles," in *2023 42nd International Symposium on Reliable Distributed Systems (SRDS)*, IEEE, 2023, pp. 151–161.
- [17] B. Romansky, T. Mazzuchi, and S. Sarkani, "Extending the update framework (tuf) for industrial control system applications," *Southeast-Con 2024*, pp. 1571–1576, 2024.
- [18] J. Samuel, N. Mathewson, J. Cappos, and R. Dingledine, "Survivable Key Compromise in Software Update Systems," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, 2010, pp. 61–72.
- [19] T. Karthik, A. Brown, S. Awwad, D. McCoy, R. Bielawski, C. Mott, S. Lauzon, A. Weimerskirch, and J. Cappos, "Uptane: Securing software updates for automobiles," in *International conference on embedded security in car*, vol. 11, 2016.
- [20] R. Lorch, D. Larraz, C. Tinelli, and O. Chowdhury, "A comprehensive, automated security analysis of the uptane automotive over-the-air update framework," in *Proceedings of the 27th International Symposium on Research in Attacks, Intrusions and Defenses*, 2024, pp. 594–612.
- [21] K. R. Jones, T.-F. Yen, S. C. Sundaramurthy, and A. G. Bardas, "Deploying android security updates: an extensive study involving manufacturers, carriers, and end users," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 551–567.
- [22] Northern.tech, "Mender: Over-the-Air Software Updates," Online. Available: <https://mender.io/>, 2026, accessed: May 8, 2026.
- [23] L.-C. Duca, A. Duca, and C. Popescu, "Ota secure update system for iot fleets," *International Journal of Advanced Networking and Applications*, vol. 13, no. 3, pp. 4988–4992, 2021.
- [24] A. K. Srivastava, C. Kirana, D. Lilaramani, R. Rakshitha, and K. Sree, "An open-source swupdate and hawkbit framework for ota updates of risc-v based resource constrained devices," in *2021 2nd International Conference on Communication, Computing and Industry 4.0 (C2I4)*, IEEE, 2021, pp. 1–6.
- [25] G. Wang, J. Chen, and C. Zheng, "Fully automated generation mechanism of rootfs for specified operating systems under linux," in *International Conference on Artificial Intelligence Security and Privacy*, Springer, 2024, pp. 12–26.
- [26] A. Tridgell and P. Mackerras, "The rsync Algorithm," Australian National University, Tech. Rep. TR-CS-96-05, Jun. 1996.
- [27] OSTree Project, "OSTree Overview," <https://ostreedev.github.io/ostree/introduction/>, 2026, accessed: Sep. 3, 2026.
- [28] B. Moran, H. Tschofenig, D. Brown, and M. Meriac, "A firmware update architecture for internet of things," RFC Editor, RFC 9019, Apr. 2021. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc9019>
- [29] Z. Zhang, G. Qin, Y. Feng, G. Wang, and K. Wang, "An improved bsdiff algorithm in automotive ota upgrade," in *Fourth International Conference on Sensors and Information Technology (ICSI 2024)*, vol. 13107. SPIE, 2024, pp. 993–1001.
- [30] H. Tan, W. Xia, X. Zou, C. Deng, Q. Liao, and Z. Gu, "The design of fast delta encoding for delta compression based storage systems," *ACM Transactions on Storage*, vol. 20, no. 4, pp. 1–30, 2024.
- [31] D. G. Korn, J. MacDonald, J. C. Mogul, and K.-P. Vo, "The vcdiff generic differencing and compression data format," RFC Editor, RFC 3284, Jun. 2002. [Online]. Available: <https://www.rfc-editor.org/rfc/rfc3284>
- [32] A. Tridgell and P. Mackerras, "The rsync algorithm," Australian National University, Tech. Rep., 1996, available: https://rsync.samba.org/tech_report/
- [33] E. van der Kouwe, D. Andriess, H. Bos, C. Giuffrida, and G. Heiser, "Benchmarking Crimes: An Emerging Threat in Systems Security," *CoRR*, vol. abs/1801.02381, 2018.
- [34] S. Jain, A. Agrawal, and N. Goyal, "Towards understanding the optimization landscape of grp0 and its variants," in *First Workshop on Foundations of Reasoning in Language Models*, 2025.