International Journal of Scientific Research & Engineering Trends
Volume 12, Issue 6, Nov-Dec-2026, ISSN (Online): 2395-566X

SPARDHA: A Mobile-First Framework for on-Device Pothole Detection, Route-Aware Alerts, and Automated Reporting of Road Hazards To Civic Authorities

Dr. Rudramurthy V C[1], T Dhanush[2], Yagnesh L[3], Vikas Reddy[4]
Department Of Computer Science And Business Systems, B. M. S. College Of Engineering, Bengaluru, Karnataka, India
{Rudramurthy.Cbs, Tdhanush.Bs23, Yagneshl.Bs23, Vikasreddy.Bs23}@Bmsce.Ac.In

Abstract- Potholes and unmarked speed breakers are making rides uncomfortable, wear vehicles out faster and create a genuine hazard for road safety, broadly speaking; indeed, manual road inspections are slow, and complaints handed in by citizens often arrive without location data or any supporting indication, in actual application. Till then, most people already carry smartphones whose motion sensors are capable of sensing road-surface challenges, in real-world scenarios. A React Native app is collecting data from the phone's accelerometer and gyroscope, splits the stream into 20 x 8 windows and runs a TensorFlow Lite model on the device to label each window as NORMAL, POTHOLE or SPEED_BREAKER, broadly speaking. Turning phone vibrations into the usable road information, though, isn't just a matter of building a classifier, like the framework must record and analyse data, attach a location to every detection, warn the rider in time, combine repeated reports of the same issue, remain usable by the public, and send summaries to the right authorities without anyone filing a complaint by hand; this paper is offering SPARDHA, a system that's tackling all of these stages are ending to end. Every trip is, indeed, typically written to a CSV file for later study or retraining, and when a pothole is detected, the rider receives an alert before getting the hazard; the detection is then stored locally in SQLite and uploaded to a server whenever the network becomes available. The server, built on FastAPI, Kafka and PostGIS, accepts a detection only when the model score is at least 0.70, merges same-type reports within 15 metres, and sends a summary of supported potholes to the responsible authorities so that repairs can be planned without waiting for complaints. The whole system is running on Google Cloud, which is making it accessible to anyone with an Android phone; in fact, testing on an Android release build confirmed that the model loads, generates real-time predictions, raises rider alerts, connects to the API, an application programming interface, an application programming interface, and filters trips by route. The paper closes by setting out the field evaluation that any claim about precision, speed or reliability will (to some degree) have to pass.

Keywords, like pothole detection; smartphone inertial sensing; on-device inference; TensorFlow Lite; Apache Kafka; CSV data collection; push alerts; Google Cloud deployment; automated civic reporting, in real-world scenarios.

I. INTRODUCTION

Roads wear out over time because of traffic, weather, utility work and delayed repairs. Potholes and sudden, unmarked speed breakers are two of the most common results; both degrade ride quality, damage vehicles and increase the danger for two-wheeler riders in particular. Checking roads manually is slow and expensive, and citizen complaints often lack location, time or photographic proof. Because most people now carry phones fitted with motion sensors, researchers have proposed using accelerometer and gyroscope data gathered during ordinary trips to spot road issues. Classifying the vibration signal, however, is only the first step; the harder challenge is managing the full life of a detection. Sensors also react to turns, braking, the way the phone is held and where it is placed. A weak GPS fix can put a problem in the wrong lane. The same pothole seen several times turns into multiple reports unless they are combined. An alert that arrives too late is of no use to the rider. A system that runs only on a developer's laptop is not a service. And even if every issue were mapped, roads would still not be repaired unless the right officials received the data in a usable form. A trustworthy system therefore has to treat data collection, analysis, warning, delivery, deployment and reporting to the authorities as one problem rather than as separate pieces.

"SPARDHA" is the framework we built around this idea. From the phone to the cloud and back to the authorities, it handles the complete chain: a React Native app collects motion data, splits it into the fixed windows and runs a TensorFlow Lite model directly on the phone; every trip is saved as a CSV file so that the raw data is remaining available for study or for sharpening the model; as soon as a pothole is detected, the rider receives an on-screen warning before getting the hazard; detected issues are stored safely on the phone and sent to a FastAPI endpoint whenever the network is allowing; the server forwards them to a Kafka stream, checks them in PostgreSQL/PostGIS and, using a confidence rule, combines same-type reports into a combined set of supported potholes;. The backend runs on Google Cloud, so anyone with an Android phone can use SPARDHA; this

© 2026 IJSRET

paper sketches SPARDHA honestly: it's explaining what we have tested so far. What's more, a ready-made report is finally received by the authorities so that repairs can be planned without waiting for complaints. As, well, what any future claim about the system will have to pass.

Verified with the work reviewed in Section 2, our advancements are normally about the framework rather than only the model:

- an blended, recorded pipeline that carries an finding from inertial sampling and CSV data collection through on-device inference, rider alerting, location gating, durable local queuing, async ingestion and geospatial retrieval;
- a strict separation between raw candidate reports and validated active oddities, regulated by configurable confidence and same-type merge rules, so that every pushed record is supported rather than assumed;
- an escalation path that closes the loop with governance, like an automated digest of confirmed potholes is delivered to the responsible authorities on a fixed schedule, converting crowdsourced detections into the doable maintenance work orders;
- a reproducibility-oriented account of the model tensor contract, feature ordering, gating thresholds, API endpoints and known shortcomings; and
- a publicly reachable rollout of the full stack on Google Cloud, together with an evaluation protocol that is fixing how event-level detection, duplicate suppression, route relevance, localisation error, latency, resource use and offline recovery must be measured before any claim is made, to a significant degree.

A working public version of the framework is already rolled out on Google Cloud, together with a plan for testing detection, duplicate removal, route relevance, location accuracy, speed, resource use and offline recovery, on the whole; the model labels are NORMAL, POTHOLE, and SPEED_BREAKER; this paper doesn't claim that the system works for cracks, ruts, edge damage or waterlogging, in many respects, and yet, that it behaves identically across the vehicles, phone placements, road types, speeds or weather; those components still have to be tested rather than suggested, in many respects.

The remainder of the paper reviews related work (Section 2), states the targets (Section 3), details the method and design (Section 4), entries implementation specifics (Section 5), reports findings together with their evidence limits (Section 6), sets out the planned ablations (Section 7), and concludes (Section 8), broadly speaking.

II. RELATED WORK

1. Road Monitoring With Phone Sensors

Sattar, Li and Chapman survey how smartphone accelerometers, gyroscopes and GPS receivers can be employed for road-surface assessment, stressing that findings rest heavily on how the data is currently collected and on how the driver behaves [1], on the whole; singh and colleagues are combining time-based vibration patterns with surface observations contributed by citizens in their Smart Patrolling system [2], in actual application. Wu and colleagues study resampling, feature construction and category for the pothole task, and warn that good results in one road environment don't automatically transfer to another [3]. As, well on work has moved the field forward in several directions, such as two-stage pavement screening [13], neural estimation of roughness from crowdsourced data [14], recognition of potholes, speed bumps and manholes [15], incremental handling of concept drift [16], a multi-sensor road-safety dataset [17], and a recent survey centered on vision-based methods [18].

The common lesson is sobering for anyone who wants to put such a system into use. The accelerometer records how the phone-vehicle system reacts, not how the pavement feels, so a score produced on the device can never by itself be a confirmed defect. SPARDHA therefore separates on-device candidate detection from everything that follows: checking the

Coordinates, confirming the result on the server and, importantly, passing the confirmed defects on to the authorities who can, to some extent, plan repairs, to a significant degree. Whether each part of the system earns its place is still an open question; the protocol in Section 6 was designed to find out, broadly speaking.

2. Cross-Condition Robustness And Vision-Based Datasets

Alam et al. study roads profiled with phones mounted on many kinds of vehicles and held in distinct positions, and find that precision measured in one setting can fall apart when phones are employed in many diverse ways [4]; this matters a great deal for a framework in which people will hold phones, keep them in pockets or mount them wherever they're liking, broadly speaking. Vision-based research, by contrast, contributes well-labelled datasets, such as RDD2022 is gathering 47,420 street photographs from six countries together with more than 55,000 labelled damage spots [5], to a significant degree; and thus, such datasets help train and test camera-based detectors, but they aren't what the present model consumes; the present model takes eight quantitative motion channels over twenty samples. What most papers in both strands ignore is, in practice, the step from spotting a obstacle to actually fixing the road - they're stopping at the classifier or the map and leave out the

© 2026 IJSRET

reporting and escalation that are deciding whether anything changes in practice. Alerting the rider, exporting raw trip data as CSV, hosting the system in the cloud and sending a ready-made summary to the authorities are all treated as first-class goals, not extras bolted onto detection; indeed, SPARDHA takes the opposite position. Adding a vision branch would mean designing a way to fuse and assess the two data modalities, not simply reusing existing image data.

3. On-Device Inference And Platform Constraints

Prior work is finding that running a model locally saves uplink bandwidth and lets category continue, in many respects; indeed, in the meantime, the older Interpreter APIs are remaining supported [7], in real-world scenarios. At the same time, the device is offline, in real-world scenarios, really, the shipped artefact uses TensorFlow Lite; the current runtime documentation is appearing, under the name LiteRT; broadly speaking; mobile inference still hinges on how preprocessing is done - the order of the input, the way values are normalised, the timing of samples and the handling of missing samples must match exactly what was used during the training. What's more, android may also limit the rate at which motion-sensor callbacks are delivered on recent versions unless the high-sampling-rate permission is declared; declaring the permission is a request rather than a guarantee, so the real arrival times of sensor events must be measured on the target hardware, broadly speaking.

4. Geospatial And Event-Stream Primitives

Turning a stream of sightings into map-grade entries needs spatial aggregation, and clearly defined event states, broadly speaking, and PostGIS provides typed geometry types and predicates such as ST_DWithin, which are working efficiently with index-backed filtering when indexes are provide [8], in many respects. What's more, kafka offers durable, replayable data streams [10]; these, though, are only building blocks, covering the application must handle anomaly identity, ensure idempotent processing and keep duplication low, in many respects, and these behaviours have to be built and showed on top of

the primitives. A scheduler and an e-mail transport can likely deliver a digest to an authority, in actual application; though, making sure the summary includes only supported data with exact coordinates and sighting counts is currently something the application itself must guarantee, in many respects, and the same holds for the reporting side, broadly speaking.

Table 1. Feature-scope comparison of the cited works and the SPARDHA prototype. 'Not documented' means the cited description does not confirm the capability; it does not prove that the capability is missing. The matrix compares feature scope, not predictive performance.

Source	Inertial path	Vision path	Crowd / spatial aggregation	Rider alert	Authority reporting
Sattar et al. survey [1]	surveyed	surveyed in context	covered	absent from the cited account	absent from the cited account
Smart Patrolling [2]	yes	not central	yes	absent from the cited account	absent from the cited account
Wu et al. [3]	yes	no	classification happens server-side	absent from the cited account	absent from the cited account
Alam et al. [4]	yes	no	event localisation across trails	absent from the cited account	absent from the cited account
RDD2022 [5]	no - image corpus only	yes - annotated imagery	outside the dataset's scope	outside the dataset's scope	outside the dataset's scope
SPARDHA (this work)	yes - 20 x 8 inertial window	no camera branch yet	raw kept apart from confirmed; proximity merge at 15 m	yes - on-screen alert on detection and along-route warnings	yes - automated digest of confirmed pothole to the respective authorities

III. OBJECTIVES

© 2026 IJSRET

Five aims were set before starting to build the prototype. Clearly, they were fixed at the outset and are normally used as the standard against which the findings in Section 6 are judged:

- Build a data-collection application that entries each pothole event together with the raw sensor readings and turns that data into CSV files, so that every trip can be audited, assessed and reused for retraining.
- Build the SPARDHA app on a Kafka- and TensorFlow-centred stack, connecting the React Native client and its on-device TensorFlow Lite classifier to a FastAPI/Kafka/Redis/PostgreSQL-PostGIS backend that ingests and checks road-hazard events, in many respects.
- Build a framework that's telling the user immediately when a hazard is spotted, using on-device inference for instant in-app warnings and a route-corridor check to announce supported hazards that are lying ahead on the planned path, in real-world scenarios.
- Roll out the app on Google Cloud and open it to everyone, so that registration, represent, confirmation and map retrieval all are happening at a public, production-grade endpoint rather than on a private development machine, on the whole.
- Make an automated report of detected potholes and send it to the authorities responsible for repairs, so that every checked defect is reaching the repair office as a clear, doable summary without anyone having to file a complaint by hand, broadly speaking.

Aims 1-4 are about building and delivering the framework; objective 5 closes the loop by turning pothole detections into the scheduled maintenance work, broadly speaking; more accurately, the split is weigh, including a prototype that can't yet prove field accuracy has to say so in the same paper that's explaining its architecture, in real-world scenarios.

IV. METHODOLOGY AND SYSTEM DESIGN

1. Architectural Overview

The pipeline is shown by figure 1 as four stages spread over two tiers. The phone senses, infers, gates and queues events (Stages 1 and 2); the server ingests, checks and stores, alongside reports (Stage 3); the map application consumes the location data (Stage 4). More accurately, every limit shown in the figure is a prototype value chosen for protection, not a value optimised against labelled data; Section 7 is recapping how each of these values would be revisited experimentally, on the whole.



Fig. 1. SPARDHA methodology: four stages across handset and server tiers, starting with on-device inertial sensing, followed by CSV-audited collection and inference, rider alerts, fault-tolerant persistence, asynchronous confirmation and automated authority reporting on Google Cloud. The final stage is route-aware retrieval. The thresholds shown are configured prototype values, not optimised ones.

2. Sensing And Window Construction

The client is built with React Native on Expo [19] [20]. Motion callbacks deliver three-axis accelerometer and gyroscope data. In this setup a gyroscope reading can be paired with the most recent accelerometer callback, which leaves the two streams out of sync unless their timestamps are aligned or both are resampled. Every captured journey, including its windows and detected events, is saved as a CSV file. The export serves two purposes: it is an audit trail for data collection, and it provides raw material for future retraining. The known timing issue is not hidden; it is documented, and the CSV export makes it easy to check the timing directly.

Each eligible window is holding twenty time steps of eight attributes - acceleration along x, y and z; rotation along x, y and z; acceleration magnitude; and rotation magnitude, to a significant degree; and so, ten samples are being advanced by the window manager at a time. The token sampling interval is 33 milliseconds, i.e. about 30 samples per second. What's more, if the gap between two consecutive samples exceeds 100 milliseconds the window is discarded or reset, in real-world scenarios; the values are stored as a [1, 20, 8] float tensor. But still, the actual sampling rate, jitter and number of dropped samples must be computed from the recorded timestamps on each device; these values are obtained from the encapsulated data, including the CSV exports, rather than assumed from setup.

3. Candidate Gating

© 2026 IJSRET

Three gates stand between a raw class score and a candidate record. Let's start up up, inference is running only when vehicle speed is 5 km/h or more, which is leaving out stationary handling of the phone, to a significant degree; moving on, a satellite fix is attached only when it's less than 10 seconds old, in actual application, and its reported horizontal precision is 20 metres or better; Android is stating that accuracy figure as an uncertainty estimate, not a bound that every individual fix obeys [21], in actual application. But still, still, the confirmation threshold of 0.70 that controls promotion later in the pipeline is an engineering safeguard, not a statistically got operating point, in many respects, and then, the class score is treated as a model score rather than a adjusted probability.

4. Offline-First Persistence And Synchronisation

To a significant degree, when a window is classified as POTHOLE or SPEED_BREAKER the rider receives a warning - an on-screen alert states the defect type, to a significant degree. And, if a route is active, the distance to the hazard ahead, so the warning arrives before the hazard rather than after it, broadly speaking, and after the warning, the event is written to SQLite with an identifier and a PENDING state before the first upload attempt, to a significant degree. The app retries uploading when it starts, when it returns to the foreground and at intervals; in the meantime, it stays open; a record becomes SYNCED only after the server admits

receipt; if a transport error or a non-successful response emerges, the record stays pending and the retry counter increases, in many respects; SQLite guarantees atomic commits [24] [25] [26]. And yet, delivery can be guaranteed by no storage system if the process is force-stopped; in turn, then again, the operating system is cutting short background work, in actual application; and thus, end-to-end fault injection is therefore still needed. As, well, a full outage-recovery demonstration hasn't yet been ran.

5. Ingestion And Confirmation

Candidates are mainly posted to /api/v1/anomalies. A candidate is being boosted by the confirmation service when its model score reaches 0.70 and merges same-type reports within 15 m, so that repeated sightings collapse into a compact set of verified active oddities; this corroborated set is feeding the reporting tier outlined below. The FastAPI service checks the submission, employs Redis-backed rate limiting as a control and publishes the event to Kafka so that reception stays decoupled from processing [10]-[12]. Both configurable values invite tuning on labelled cases: a radius that's too extensive can fuse two separate defects into one pin. At the same time, one that's too tight can split a single defect across several pins. What's more, the merge isn't supposed to be a full spatio-temporal clustering method unless the code is shown to do one. The current figures are introduced as arrangement, not optimisation.

6. Route-Aware Retrieval

Three read APIs keep their finding sets separate on purpose; GET /api/v1/anomalies/nearby returns checked hazards close to a given point; GET /api/v1/anomalies/raw/nearby returns raw sightings, which must never be showed on the map as confirmed defects; and POST

A supplied geometry is mainly taken by /api/v1/anomalies/along-route and filters the backed up hazards that fall inside a 15-metre corridor, to a significant degree. Route coordinates are sent longitude-first while the nearby factors are employing latitude-first; the code calls out this asymmetry clearly, because mixing up the order creates real geospatial mismatches, on the whole. The along-route result set drives the alert stream - hazards checked to lie on the planned path are, to be sure, announced before the rider reaches them; while merely nearby hazards never trigger a route warning, in actual application. Distance semantics matter as well, including metre-based findings need either a geography predicate or a projected coordinate reference system, to a significant degree. And the PostGIS documentation explains the unit difference for ST_DWithin [8].

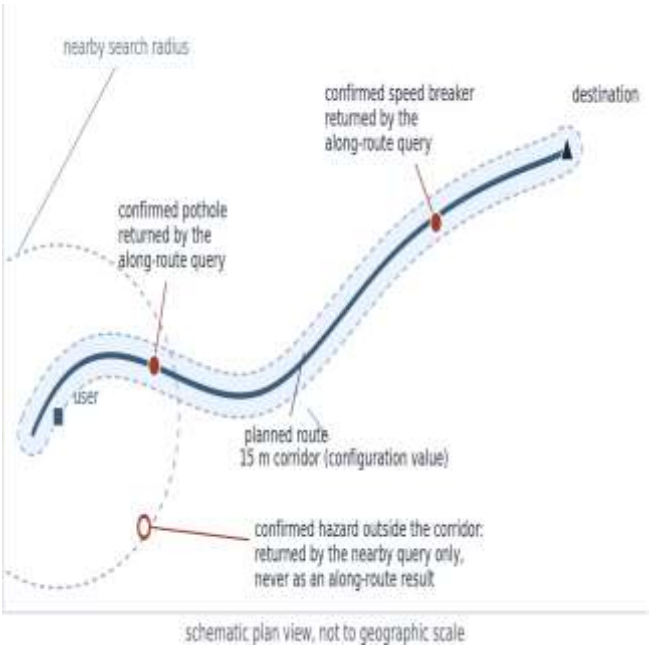


Fig. 2. Route-corridor filtering. Verified hazards inside the 15-metre corridor of the planned path are returned as along-route results and announced as ahead-of-route alerts. A hazard that is merely close by is handled by the nearby query

© 2026 IJSRET

and is never shown as being on the route. The corridor width is a configurable setting, not a survey-derived measurement.

Figure 2 is often showing why the difference matters, really, a hazard may sit close to the traveller yet far from the chosen path; treating it as along-route would hand the user misleading information and cause false alerts, to a significant degree. But GPS error can (to a meaningful extent) push a real hazard outside a tight corridor and generate a false negative, in real-world scenarios; the corridor size therefore is needing to be tested, across several values instead of counting on a single setting, in many respects.

7. Automated Reporting To Authorities And Cloud Deployment

Checked oddities are turned into the civic action by the reporting tier. A scheduled job on the server is generally gathering a report of every supported pothole, grouped by the municipal zone in which each coordinate falls, listing the sighting count, the latest corroboration and the coordinates of each pothole, and sends the digest to the responsible authorities by e-mail on a fixed schedule, on the whole. A municipal engineer so receives a work list instead of a flood of individual complaints, on the whole; indeed, as, well, every entry is keeping its coordinates and sighting count so that prioritisation can be evidence-based, in actual application; in fact, two design rules regulate the report, like only checked data are covered, raw sightings are never used, in real-world scenarios.

The backend runs on Google Cloud. The FastAPI service, Kafka, Redis and the PostgreSQL/PostGIS store all run on Google Cloud Run behind managed infrastructure and are reachable at a public endpoint; the same deployment handles registration, represent,

confirmation and map retrieval for every user. Running the stack on managed cloud infrastructure instead of a development tunnel turns the prototype into an actual service that anyone can (in practice) install and use, on the whole; as, well, it lets the automated authority reports leave on schedule from a machine that never goes offline, in many respects.

V. IMPLEMENTATION DETAILS

1. Model And Tensor Contract

The runtime contract of the shipped artefact is, for all concrete purposes, being recorded by table 2, broadly speaking. Two points deserve emphasis. To start, feature scale and ordering directly alter inference, so the normalisation configuration is stored together with the model, in many respects; what's more, the paper-ready release must also report how each channel is standardised; sign, axis and unit drift are inspected against fixed raw-window fixtures that are encompassing projected tensor values, to a significant degree. Then, the CSV exports created by the data-collection flow naturally (or more precisely) carry these fixtures - each journey file holds the eight-channel windows in the exact order the model consumes, so a fixture is simply a row range from a real encapsulate; in many respects. Also, so, the training provenance of the model doesn't appear anywhere in the existing implementation record. What's more, this paper intentionally doesn't link the artefact to RDD2022 or any other named corpus; the model authors must supply provenance, split shows and normalisation constants before any training-data claim can be made, really, also provenance must never be guessed from a filename; in fact, how about this, the CSV journey exports are the process by which any future labelled corpus will be built, which is why the collection objective treats them as a first-class output rather than a debugging convenience, in actual application.

Table 2. Model and runtime contract documented for the SPARDHA prototype. The digest must be recomputed from the final artefact at submission time.

Item	Documented configuration
Runtime format	TensorFlow Lite (.tflite)
Input tensor	[1, 20, 8] float32; one batch, twenty time steps, eight ordered channels
Output tensor	[1, 3] float32; fixed class order: NORMAL, POTHOLE, SPEED_BREAKER
Feature order	a_x, a_y, a_z, g_x, g_y, g_z, |a|, |g|
Window parameters	twenty samples per window, advance of ten, nominal 33 ms spacing; reset whenever a gap exceeds 100 ms
Data export	each captured journey written to a CSV file with the same ordered channels, serving audit and retraining
Artefact identity	227,288 bytes; SHA-256 FF6E38D69B9EEA5BAC7CF971A69911BD55345FC2C3EC2F5D2C3CCAACABF0997E
Training provenance	not established in the available implementation record; to be supplied by the model authors and never inferred from the filename

© 2026 IJSRET

2. Backend Stack And Api Contracts

On the server side, FastAPI handles HTTP, PostgreSQL broadened with PostGIS spatially stocks inside a Docker-managed service, Kafka carries event ingestion, Redis is backing the rate limiter, in many respects, and raw findings are, in fact, deliberately respected from confirmed active exceptions by the read APIs. As, well, the corridor filter server-side is being used by the along-route endpoint; event schemas and acknowledgements should be versioned, in actual application. What's more, a scheduled reporting job compiles the automated authority digests, in many respects; what's more, clients should carry idempotency keys so that retries can't create duplicate server records; for distribution, these contracts belong in a versioned API appendix, on the whole. The whole tier is rolled out on Google Cloud, with the API service running as a containerised workload on Cloud Run behind a public HTTPS endpoint [27], so registration, represent, confirmation, retrieval and reporting all run on infrastructure reachable by every user rather than only from a development machine.

Table 3. Documented API and coordinate semantics. The final release should attach request and response schemas or a versioned API appendix.

Endpoint / job	Purpose	Coordinate / state semantics
POST /api/v1/anomalies	Accept a candidate observation	keeping the schema versioned; the client supplies a stable idempotency key
GET /api/v1/anomalies/nearby	Confirmed, active hazards near a point	latitude and longitude plus a radius; only confirmed and active records qualify
GET /api/v1/anomalies/raw/nearby	Raw sightings near a point	unconfirmed sightings never appear on the map as verified defects
POST /api/v1/anomalies/along-route	Confirmed hazards close to a supplied path	each route point ordered by longitude and latitude; corridor fixed at 15 metres by configuration
Scheduled reporting job	Compile and dispatch the authority digest	grouping confirmed potholes by zone; sighting counts and coordinates retained; raw sightings never included

3. Platform Constraints

As, motion-sensor delivery might be delayed by newer Android versions unless an app specifically is asking for it the build manifest includes the HIGH_SAMPLING_RATE_SE NSORS permission [6], in many respects. Logs from the release build show the listener is registering with a zero-microsecond delay, which means it's asking for data quickly as the hardware can offer it instead of getting a steady 30 Hz stream, on the whole; indeed, before any claims about data quality are made, the time between samples the rate of missing packets and the timing of callbacks must be inspected for each device, in real-world scenarios. The CSV files are available, for this verification process using real data.

4. Deployment And Security Posture

The cloud deployment replaces the development tunnel used by earlier builds - it serves the endpoint over HTTPS with managed TLS and handles scaling, restarts and logging, in many respects. Reliability, transparency, privacy and operational risk can be worked through the systematically with the NIST AI Risk Management Framework, which is recommending a management process rather than certifying that an application is safe or obedient [22]. Production authentication, secrets management, strict rate limits and audit logging all are remaining to be used, and retention and access rules for findings including location data must be published before the service is advertised beyond the pilot group. The application itself must still be hardened. What's more, the authority-reporting channel deserves the same discipline, like recipient lists must be verified; so, what's more, the digest must never carry raw sightings or personal data, in many respects. Clearly map-marker and offline-sync patches entered the source tree after the build that was tested, and the patched release APK has yet to undergo regression validation, broadly speaking.

VI. RESULTS AND DISCUSSION

1. Scope Of The Evidence

These outcomes below are development-integration results, not a field trial;. As well, so, the project record is suggesting that the recorded paths run in an Android release build and against the cloud-deployed backend, in real-world scenarios, but still, what it includes is no labelled held-out set and no multi-device campaign, and therefore no total precision, recall or calibration figures, to a significant degree; some database rows use test coordinates and aren't road ground truth; in turn, stating this boundary up front is precisely what makes the positive findings meaningful, in many respects.

2. On-Device Inference

© 2026 IJSRET

Runtime logs are showing that the TFLite model loads correctly in the release build and projects while live sessions are running, to a significant degree; and so, table 4 lists representative examples, namely a NORMAL window scored about 0.996, and plus, on top of that, two separate SPEED_BREAKER windows scored 0.604 and 0.491, on the whole. Every POTHOLE window classified during the sessions triggered a rider alert on the handset, to a significant degree; indeed, no reading below 0.70 was ever encouraged anywhere in the flow, which is matching the behaviour anticipated from the arranged gate. As, well, no total classification performance can be inferred from them; as, well, what every recorded journey created was a CSV file whose columns matched the recorded feature order. As, the logged windows are usually few and weren't sampled in a controlled way, they're diagnostics rather than an evaluation sample, to a significant degree.

Table 4. Representative model scores seen in Android release-build logs. Diagnostic examples only; the logged windows are not a controlled sample, so no accuracy, recall or calibration figure is derived from them.

Log observation	Predicted class	Model score	Rider alert / promotion
Window A	NORMAL	0.996	No alert; no event created
Window B	SPEED_ BREAKER	0.604	Alert raised on the handset; not promoted (under the 0.70 bar)
Window C	SPEED_ BREAKER	0.491	Alert raised on the handset; not promoted (under the 0.70 bar)

3. Service Integration Checks

Table 5 summarises each completed check together with what it does and doesn't establish, in real-world scenarios; and so, the OpenAPI document returned HTTP 200 from the public cloud endpoint, which is showing that the service is publicly reachable but doesn't prove uptime or the ability to handle traffic, in real-world scenarios.

Both findings are matching the arranged settings, and yet, neither is saying anything about localisation accuracy or route-level recall. The retry patch is in place with the triggers outlined; and thus, the reporting job gathered a digest of validated oddities, grouped the data as anticipated and delivered it to a test authority mailbox on schedule. The along-route query returned a test anomaly when the geometry crossed its location, and in a separate run returned zero hazards when the test point lay outside the 15-metre corridor, and only the same, no full reconciliation has been recorded-covering what happens when a connection is lost and later restored, acknowledgements and forced-stop behaviour - so zero-loss delivery is not claimed in this section, on the whole.

Table 5. Integration checks completed at manuscript time, with the evidence limits of each.

Check	Outcome in the record	Not established
TFLite initialisation (Android release)	confirmed by log output	predictive accuracy or robustness of the model
Live on-device predictions	sample scores logged (Table 4)	per-class precision, recall, F1 or calibration
Rider alert on detection	on-screen alert raised for each classified hazard window in live sessions	alert lead distance or distraction effects at speed
CSV journey export	one file per capture with the documented feature order	completeness under force-stop or storage exhaustion
Cloud deployment (Google Cloud)	public HTTPS endpoint served the OpenAPI document with HTTP 200	uptime, load behaviour or regional failover at scale
API reachability	OpenAPI returned HTTP 200 from the public cloud endpoint	security posture under adversarial traffic
Along-route corridor query	returned an aligned test event; excluded a misaligned one in a separate run	localisation accuracy or route-level recall
Automated authority report	digest of confirmed anomalies compiled and dispatched on schedule to a test mailbox	action taken by the receiving authority; digest precision in the field
Pending-event retry	patch installed with documented triggers; full outage matrix not recorded	zero-loss delivery or guaranteed background execution

© 2026 IJSRET

Check	Outcome in the record	Not established
Map marker / destination patch	patch in source; regression testing of the patched build pending	packaged UI correctness on other devices
Ground-truthed field evaluation	no such campaign recorded	any numerical real-world performance claim

4. Application Interface

Fig. 3 shows the Android client. The monitoring home screen (a) displays location and speed alongside the anomaly heatmap and journey planning. The route view (b) draws the planned route as a polyline with hazard markers for the confirmed anomalies that lie along it, a banner telling the rider how many anomalies lie ahead, and a route summary card listing the distance, the estimated time and the on-route anomaly count. An anomaly summary card lets the rider compare alternatives before starting navigation. Nearby and along-route results are kept in logically separate sets, so interface totals never mix local context with route risk.

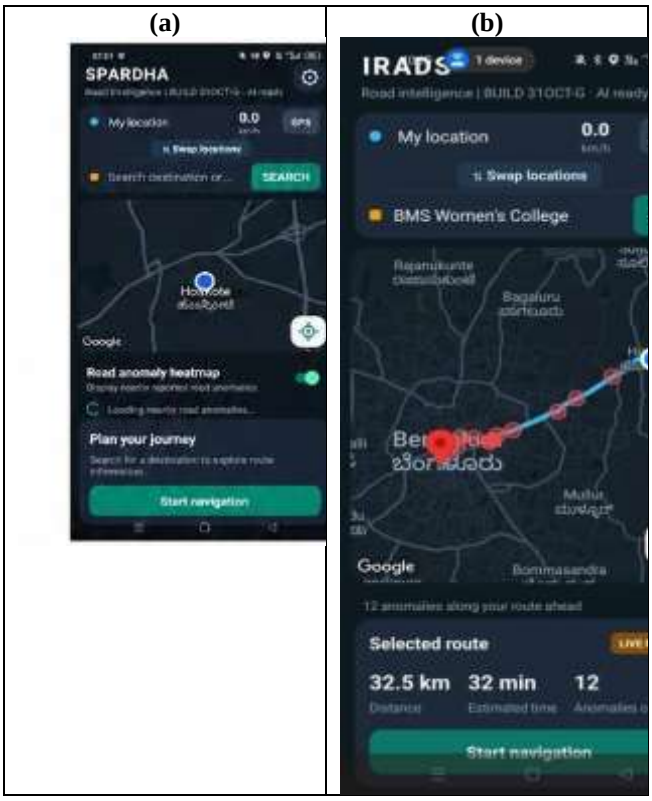


Fig. 3. SPARDHA client on Android: (a) monitoring home screen displaying location, anomaly heatmap and journey planning; (b) route view showing the planned polyline, typed hazard markers from the along-route query, an anomaly banner and a route summary card reporting distance, estimated time and the on-route anomaly count.

5. Open Measurements And Protocol

We haven't yet measured end-to-end latency with a tough test harness, really, a sound use would timestamp every stage separately, namely when the sensor completes a window, when Preprocessing finishes, when TFLite inference runs, when an event is saved, when the alert appears, how long a request to the cloud API takes, when the server checks the event and when the map is drawn; it would report the median, the 95th percentile and the maximum for each step over several runs, and record the phone model, OS build, thermal state and network condition, with battery level and memory usage logged under a fixed rule, in real-world scenarios. In plain words logging timestamps doesn't generate a measurement; the start and stop points and the number of samples must be prescribed first, in many respects.

Generalisation also is remaining open. Future checks must, by design, split data by session and by route

- ideally keeping whole phones or vehicle placements separate - cover several speeds, include sections that are deliberately rough but free of reportable defects as well as speed breakers, manholes and expansion joints where present, on the whole. Also, repeat passes over every defect. The authority-reporting loop adds one further outcome measure, the proportion of digest entries that are corresponding to genuinely repairable defects, which can only be established jointly with the authorities that receive the reports, broadly speaking. The metrics to report are per-class precision, recall, and F1, macro-F1, the confusion matrix, event-level precision and recall, alerts per kilometre travelled, alert lead distance and the localisation offset of each verified pin - all computed from held-out labels, since randomly splitting overlapping windows would leak data, namely neighbouring windows from a single drive share most of their samples [23].

VII. PLANNED ABLATIONS AND EVALUATION ROADMAP

No ablation has been run to date. Removing a feature channel needs either a retrained model or a carefully controlled input ablation - malformed tensors must never be fed to the current model, in actual application. Table 6 sets out the matrix that would separate the value of each guard; versions are inspected on held-out routes and notes, with the model and all thresholds locked before testing begins, in real-world scenarios, really, the 'Hypothesis' column lists directions to study, not findings, broadly speaking.

© 2026 IJSRET

Table 6. Planned ablation matrix. No cell is filled until the matching experiment has been carried out; the hypothesis column lists directions to examine, not results.

Variant	Change from the full pipeline	Primary outcomes	Hypothesis
A. Full prototype	every current gate and confirmation rule active	macro-F1; event precision/recall; localisation; duplicates	reference configuration for comparison
B. No confidence threshold	candidates promoted at lower scores	false alerts per km; precision and recall	recall may rise while precision falls
C. No GPS quality gate	fixes accepted regardless of age or accuracy	median and P90 location error; invalid-location rate	location error may grow
D. No spatial merge	the 15 m same-type merge switched off	duplicate rate; unique-defect count error	map and digest entries may multiply
E. No route corridor	nearby list returned without path-distance filtering	along-route precision and recall; irrelevant alerts	route relevance and alert precision may degrade
F. No offline queue	uploads attempted with no lasting pending state	event loss; recovery time; duplicate submissions	outages may lose events entirely
G. No rider alert	detections saved but never surfaced to the rider	alert lead distance; rider-reported usefulness	warning value is unmeasured without the channel
H. Accelerometer-only features	gyroscope channels dropped via compatible retraining or controlled input ablation	macro-F1; robustness to device and placement	the value of rotation channels is uncertain

The roadmap is deliberately ordered: first establish training provenance and preprocessing constants, then gather route- and session-labelled ground truth, then lock the protocol, and only then fill in the matrix, to a significant degree; indication must live in entries kept for that purpose; it doesn't come automatically just, because the software stack exists [22] [23], broadly speaking.

VIII. CONCLUSION

SPARDHA proposes a phone, a classifier and a disciplined cloud backend can be combined into a road-health service in which every step is visible and testable, namely CSV-audited data collection, sensing gates, immediate rider alerts, offline-friendly retention, asynchronous confirmation with adjustable merging, retrieval that's following route shape rather than mere distance, a public Google Cloud deployment, and an automated reporting loop that delivers checked potholes to the repair authorities, in actual application. The findings established so far are integration results - model start-up and live predictions in an Android release build, on-screen alerts on detection, CSV journey exports in the recorded feature order, public access to the cloud endpoint, corridor filtering that admitted a correctly matched up test anomaly while rejecting a misaligned one, and a scheduled authority digest delivered in the anticipated format, and are honest but incomplete, including this paper is making no claim about field accuracy, latency, cross-device stability, location error, alert lead time, repair outcomes or guaranteed offline delivery, because the supporting proof hasn't yet been collected, to a significant degree. The advancement is the recorded framework together with the reproducible protocol that any stronger claim must now meet; completing that check is the immediate next step for this work, in real-world scenarios.

1. Acknowledgment
The authors thank the Department of Computer Science and Business Systems at B. M. S. College of Engineering, Bengaluru, for the resources and guidance that supported this project, in actual application.

© 2026 IJSRET

REFERENCES

1. S. Sattar, S. Li, and M. Chapman, "Road Surface Monitoring Using Smartphone Sensors: A Review," Sensors, vol. 18, no. 11, art. no. 3845, 2018. doi: 10.3390/s18113845.
2. G. Singh, D. Bansal, S. Sofat, and N. Aggarwal, "Smart patrolling: An efficient road surface monitoring using smartphone sensors and crowdsourcing," Pervasive and Mobile Computing, vol. 40, pp. 71-88, 2017. doi: 10.1016/j.pmcj.2017.06.002.
3. C. Wu, Z. Wang, S. Hu, J. Lepine, X. Na, D. Ainalis, and M. Stettler, "An Automated Machine-Learning Approach for Road Pothole Detection Using Smartphone Sensor Data," Sensors, vol. 20, no. 19, art. no. 5564, 2020. doi: 10.3390/s20195564.
4. M. Y. Alam, A. Nandi, A. Kumar, S. Saha, M. Saha, S. Nandi, and S. Chakraborty, "Crowdsourcing from the True crowd: Device, vehicle, road-surface and driving independent road profiling from smartphone sensors," Pervasive and Mobile Computing, vol. 61, art. no. 101103, 2020. doi: 10.1016/j.pmcj.2019.101103.
5. D. Arya et al., "RDD2022: A multi-national image dataset for automatic road damage detection," Geoscience Data Journal, vol. 11, no. 4, pp. 846-862, 2024. doi: 10.1002/gdj3.260.
6. Android Developers, "Sensors Overview," Android Developers documentation. Available: https://developer.android.com/develop/sensors-and-location/sensors/sensors_overview. Accessed: 24 Oct. 2026.
7. Google AI Edge, "Getting started with LiteRT," Google for Developers. Available: https://developers.google.com/edge/litert/overview. Accessed: 24 Oct. 2026.
8. PostGIS Project, "ST_DWithin," PostGIS documentation. Available: https://postgis.net/docs/ST_DWithin.html. Accessed: 24 Oct. 2026.
9. Project OSRM, "OSRM API Documentation," Open Source Routing Machine documentation, v5.24.0. Available: https://project-osrm.org/docs/v5.24.0/api/. Accessed: 24 Oct. 2026.
10. Apache Software Foundation, "Apache Kafka Documentation." Available: https://kafka.apache.org/documentation/. Accessed: 24 Oct. 2026.
11. FastAPI, "FastAPI Documentation." Available: https://fastapi.tiangolo.com/. Accessed: 24 Oct. 2026.
12. Redis, "Rate limiter," Redis documentation. Available: https://redis.io/docs/latest/develop/use-cases/rate-limiter/. Accessed: 24 Oct. 2026.
13. K. Zhao, S. Xu, J. Loney, A. Visentin, and Z. Li, "Road pavement health monitoring system using smartphone sensing with a two-stage machine learning model," Automation in Construction, vol. 167, art. no. 105664, 2024. doi: 10.1016/j.autcon.2024.105664.
14. F. Jalili, S. M. Ghavami, and H. Afsharnia, "An artificial neural network approach to assess road roughness using smartphone-based crowdsourcing data," Engineering Applications of Artificial Intelligence, vol. 138, pt. A, art. no. 109308, 2024. doi: 10.1016/j.engappai.2024.109308.
15. G. Kim and S. Kim, "A Road Defect Detection System Using Smartphones," Sensors, vol. 24, no. 7, art. no. 2099, 2024. doi: 10.3390/s24072099.
16. I. Ferjani and S. A. Alsaif, "Dynamic Road Anomaly Detection: Harnessing Smartphone Accelerometer Data with Incremental Concept Drift Detection and Classification," Sensors, vol. 24, no. 24, art. no. 8112, 2024. doi: 10.3390/s24248112.
17. A. Khandakar, D. G. Michelson, M. Naznine, A. Salam, M. Nahiduzzaman, K. M. Khan, P. N. Suganthan, M. A. Ayari, H. Menouar, and J. Haider, "Harnessing Smartphone Sensors for Enhanced Road Safety: A Comprehensive Dataset and Review," Scientific Data, vol. 12, art. no. 418, 2025. doi: 10.1038/s41597-024-04193-0.
18. S. H. Abdelwahed, B. K. Sharobim, B. Wasfey, et al., "Advancements in real-time road damage detection: a comprehensive survey of methodologies and datasets," Journal of Real-Time Image Processing, vol. 22, art. no. 137, 2025. doi: 10.1007/s11554-025-01683-1.
19. React Native, "React Native Documentation." Available: https://reactnative.dev/. Accessed: 24 Oct. 2026.
20. Expo, "Expo Documentation." Available: https://docs.expo.dev/. Accessed: 24 Oct. 2026.
21. Android Developers, "Location API reference: Location.getAccuracy." Available: https://developer.android.com/reference/android/location/Location. Accessed: 24 Oct. 2026.
22. E. Tabassi, "Artificial Intelligence Risk Management Framework (AI RMF 1.0)," NIST AI 100-1, National Institute of Standards and Technology, Gaithersburg, MD, Jan. 2023. doi: 10.6028/NIST.AI.100-1.
23. Scikit -learn Developers, "Model evaluation: quantifying the quality of predictions," scikit-learn documentation. Available: https://scikit-learn.org/stable/modules/model_evaluation.html. Accessed: 24 Oct. 2026.
24. SQLite, "Atomic Commit in SQLite," SQLite documentation. Available: https://www.sqlite.org/atomiccommit.html. Accessed: 24 Oct. 2026.
25. Expo, "SQLite," Expo SDK documentation. Available: https://docs.expo.dev/versions/latest/sdk/sqlite/. Accessed: 24 Oct. 2026.

© 2026 IJSRET

26. React Native, "AppState," React Native documentation. Available: https://reactnative.dev/docs/appstate. Accessed: 24 Oct. 2026.
27. Google Cloud, "Google Cloud Run documentation." Available: https://cloud.google.com/run/docs. Accessed: 24 Oct. 2026.

© 2026 IJSRET