International Journal of Science, Strategic Management and Technology
Volume **02** Issue **04** April-**2026** | ISSN: **3108-1762 (Online)** | Impact Factor: 3.8
An International, Peer-Reviewed, Open Access Scholarly Journal Indexed in recognized academic databases

SurveilX: AI and IoT-Based Drone Surveillance System for Industrial Pipeline Inspection

Under the Guidance of
Mrs. Sharda Mungale
Assistant Professor, Department of Industrial IoT

Kanishk Khobragade (0133) **Om Deulkar (0139)*** **Varun Bansod (0141)** **Anurag Likhar (0126)**
Department of Industrial IoT
Priyadarshini College of Engineering, Nagpur, India
** Presenting Author*

https://doi.org/10.55041/ijsmt.v2i4.222

Cite this Article: Mungale, S. (2026). SurveilX: AI and IoT-Based Drone Surveillance System for Industrial Pipeline Inspection. International Journal of Science, Strategic Management and Technology, *02*(04). https://doi.org/10.55041/ijsmt.v2i4.222

Abstract — Pipeline inspection in industrial settings is one of those problems that sounds straightforward until you actually try to solve it. You have kilometres of pipe running through refineries and chemical plants, most of it in places where sending a person is either dangerous or just not practical. Our project, SurveilX, came out of thinking seriously about this gap. We built a drone-based system that combines IoT sensors, a small onboard computer, and machine learning to inspect pipelines and flag faults without putting anyone at risk. The drone carries five sensors — for gas, vibration, humidity, distance, and GPS location — plus a camera. All the processing happens onboard using a Raspberry Pi 4 running TensorFlow Lite models and OpenCV, so the system does not depend on a cloud connection to make decisions. An ESP32 handles the lower-level sensor readings and sends the data up to the Pi. When the system spots something — corrosion, a crack, a gas leak — it sends an alert with GPS coordinates to a live dashboard. In our tests, we hit 91% classification accuracy and an average alert time of about 3.2 seconds. We think this kind of approach can genuinely change how industrial inspection is done.

Keywords — UAV-based Inspection, IoT Sensing, Edge AI, Computer Vision, Pipeline Fault Detection, Embedded Machine Learning

I. INTRODUCTION

Industrial pipelines are everywhere—running under refineries, through chemical plants, across water treatment facilities. Most people never think about them, but they carry enormous responsibility. When they fail, the consequences can be severe: spills, fires, injuries, even fatalities. The challenge is that keeping them in good condition requires regular inspection, and a lot of that infrastructure sits in places that are genuinely dangerous for people to enter.

Right now, most inspection is still done manually. Teams of engineers walk the lines, check readings on fixed sensors, or use handheld ultrasonic gauges to look for wall thinning. The problem is obvious once you say it out loud: you cannot be everywhere at once, and between inspection visits, a fault that started small can grow into something serious. Fixed sensors help but they only

watch one spot. And sending people into confined spaces full of combustible gas is never a great idea.

When we started thinking about SurveilX, three things gave us confidence that a drone-based solution was actually feasible right now. First, affordable quadcopters have gotten good enough to carry a useful payload for ten to fifteen minutes on a charge. Second, a microcontroller like the ESP32 can handle multiple sensor streams simultaneously on a board the size of a business card. Third, quantized neural networks can now run inference on an ARM processor fast enough to be genuinely useful in the field. None of these were true even five years ago.

So we designed SurveilX to bring all three together. The drone carries a set of sensors covering the main failure modes we care about—gas leaks, mechanical stress, environmental conditions, and physical obstacles — plus a camera for visual inspection. Everything is processed onboard. Results go to a dashboard. The rest of this paper explains exactly how we built it and what we found when we tested it.

II. RELATED WORK

Looking at existing work helped us understand what had already been tried and where the gaps were. The study in [1] is a good example of what fixed IoT systems can do well — the authors combined vibration and temperature data from mounted sensors and got solid results for identifying bearing faults and overheating in rotating machinery. But the limitation is baked in: those sensors only know what is happening at the exact spot they are bolted to. Move the problem two meters down the pipe, and they miss it entirely.

The work in [2] took a different approach and put a camera on a drone, which is closer to what we were going for. They ran their deep learning model on a remote server and used the UAV just for image collection. It worked reasonably well in controlled conditions, but in the field, connectivity is unpredictable. If you are inspecting a remote stretch of pipeline and the network drops out, the whole system stalls. That is a fundamental problem with cloud-dependent designs.

Reference [3] focused specifically on leak detection using pressure and flow-rate data from fixed meters along a pipe. The sensitivity numbers were impressive, but again, fixed infrastructure means fixed coverage. There is no way to redeploy those sensors when you need them somewhere else, and the system gives you no visual confirmation of what it found.

What we took from all three was this: sensor fusion works, aerial mobility is valuable, and onboard processing is not optional if you want something that works reliably in the field. SurveilX tries to combine the best of each approach rather than picking just one.

III. PROBLEM STATEMENT

The core problem we set out to solve is that current industrial inspection methods force a trade-off between coverage and safety that should not have to exist. Specifically, we identified five tensions that any real solution needs to address:

- Keeping workers safe versus actually inspecting thoroughly — the most hazardous sections are usually the ones most in need of attention.
- Continuous monitoring versus being able to move sensors where they are needed — fixed installations cannot adapt when the situation changes.
- Catching faults early versus fitting into scheduled maintenance windows — faults do not wait for the calendar.
- Getting consistent results versus relying on human judgment under difficult conditions—fatigue and stress affect what people notice.
- Fast response times versus depending on cloud connectivity—if a gas leak is forming, waiting fifteen seconds for a cloud round-trip is too long.

A solution that only fixes some of these is not really a solution. SurveilX was designed from the start to handle all five, which is why the onboard processing architecture was non-negotiable for us.

IV. SYSTEM OBJECTIVES

Before we got into hardware selection or software design, we wrote down what the system actually had to achieve. These targets shaped every decision we made later:

- Keep the total payload under 600 g so the drone can stay airborne long enough to complete a useful inspection sweep.

- Cover at least four different physical phenomena with sensors — relying on just one type of measurement leaves obvious blind spots.
- Run all classification and anomaly detection onboard. No cloud dependency in the critical path.
- Hit fault classification accuracy above 88% for corrosion, cracking, and active leaks on our test dataset.
- Get a confirmed alert to the operator dashboard within five seconds of detecting a fault.
- Record GPS coordinates for every alert, accurate to within five metres, so maintenance teams can find the exact location afterwards.

V. SYSTEM ARCHITECTURE

We split the system into four layers, each with a clear job and clean interfaces to the others. The main reason for doing it this way was flexibility — if we want to swap out the camera or upgrade the ML model later, we can do that without touching the communication or visualisation layers.

A. Sensor Acquisition Layer

The ESP32 sits at the bottom of the stack and handles everything that touches the physical sensors directly. It reads the MQ-2 gas sensor through its 12-bit ADC, catches vibration events from the SW-420 on a hardware interrupt, pulls temperature and humidity from the DHT22 over its single-wire protocol, measures obstacle distance by timing the HC-SR04 echo pulse, and listens to NMEA sentences coming in from the NEO-6M GPS on a dedicated UART. We poll everything at 5 Hz and pack each reading cycle into a JSON object that gets sent up to the Raspberry Pi over USB serial at 115,200 baud. The ESP32 was a natural choice for this role — it handles the timing-sensitive parts well and keeps that complexity away from the Pi.

B. Edge Processing Layer

The Raspberry Pi 4 Model B is where the actual thinking happens. We run two threads in parallel: one handling the camera at ten frames per second and running each frame through our TensorFlow Lite model, and one parsing the incoming sensor JSON from the ESP32 and running the anomaly detection logic. Keeping them as separate threads was important because camera inference is bursty — it would hold up sensor processing if they ran sequentially.

The vision model is MobileNetV2 fine-tuned on about 4,200 pipeline images we labelled ourselves, covering oxidation, fractures, normal surfaces, and vapour releases. After training, we quantised it to INT8 using the TF Lite converter. That cut the model size from 14 MB down to 3.8 MB and pushed inference speed from 4 to 10 FPS on the Pi — a bigger improvement than we honestly expected. Anomalous frames get bounding boxes and confidence scores drawn on them with OpenCV before being queued for sending.

For sensor anomaly detection, we use a rolling 30-sample window per channel. Before each flight, the system runs a 60-second calibration pass on the ground and records baseline mean and standard deviation for each sensor. During the flight, any reading that sits more than three standard deviations outside that baseline for two consecutive samples triggers a hard alert. The two-sample requirement cuts down on false alarms from vibration spikes during motor operation, which was a real nuisance in early testing.

C. Communication Layer

We run an Eclipse Mosquitto MQTT broker on a ground-station laptop. The Raspberry Pi connects to it over 2.4 GHz Wi-Fi and publishes to a two-tier topic structure: raw sensor telemetry goes to surveilx/sensors/<channel> and confirmed fault events go to surveilx/alerts/<fault-class>. Each fault event payload includes the annotated frame, the GPS fix, a sensor snapshot, and a severity level. We also run a small Flask server on the Pi that exposes a REST endpoint for the dashboard to pull the latest annotated frame on demand. Separating the two protocols was a deliberate choice — mixing high-frequency telemetry and large image payloads on the same MQTT connection caused congestion problems early on.

D. Visualisation Layer

The operator dashboard runs in Node-RED on the ground station. It subscribes to all the MQTT topics and maps them to widgets: rolling time-series charts for each sensor, a world map with colour-coded fault pins (green for normal, amber for soft threshold, red for confirmed fault), an alert log, and a live video feed refreshing at about two frames per second. Every hard alert also gets

written to a local SQLite database so we have a full record of the mission to review after landing.

VI. HARDWARE COMPONENTS

Table I lists the hardware we settled on after several rounds of prototyping. A few of the choices deserve a brief explanation.

TABLE I. SURVEILX HARDWARE MANIFEST

Component	Role & Key Specification
Raspberry Pi 4B (4 GB)	Edge AI host; quad-core Cortex-A72, 4 GB LPDDR4
ESP32 Dev Module	Sensor aggregator; 240 MHz dual-core, Wi-Fi + BT
MQ-2 Gas Sensor	Combustible-gas detection; 0–5 V analogue output
SW-420 Vibration	Mechanical anomaly; configurable digital threshold
DHT22	Humidity & temp; ±2% RH, ±0.5°C accuracy
HC-SR04 Ultrasonic	Obstacle avoidance; 2 cm–4 m range, ±3 mm
NEO-6M GPS	Geo-tagging; 1–5 Hz fix, ~2.5 m CEP
Pi Camera V2	Visual inspection; 8 MP Sony IMX219, 1080p30
F450 Quadcopter	Aerial host; 450 mm diagonal, ~800 g payload
3S LiPo 2200 mAh	Power supply; ~13 min hover endurance

We went with the F450 frame partly because it is cheap and widely available, but mainly because its central mounting plate gave us room to fit the Pi, the ESP32 breakout, and the sensor cluster without needing custom brackets. The 2200 mAh 3S LiPo gets us about thirteen minutes in the air at our loaded take-off weight of around 1.1 kg, which was enough to cover the test segments we were working with. Battery endurance is the obvious next thing to improve.

VII. SOFTWARE ARCHITECTURE

Everything onboard is written in Python 3.9. We went through a few different threading models early on before landing on three daemon threads — sensor ingestion, visual inference, and network egress — communicating through thread-safe queues. The queue-based approach meant we did not have to think hard about locking shared state, and it made it easy to tune buffer sizes when we ran into latency issues during early flight tests.

A. Visual Fault Classifier

We chose MobileNetV2 as the base model mostly for practical reasons: it runs well on ARM, there is solid documentation for deploying it with TF Lite, and the ImageNet pre-trained weights gave us a good starting point even though pipeline images are very different from everyday photos. We only unfroze the last three convolutional blocks and the classification head during fine-tuning, which kept training fast and avoided the catastrophic forgetting issue we hit when we tried unfreezing the whole network.

Our dataset of 4,200 images was labelled by hand by the four of us, which was tedious but gave us confidence in the label quality. We augmented it heavily — random rotations, HSV colour shifts, simulated lens flare — mostly to cover the lighting variation you get when flying at different angles and times of day. Training ran for 50 epochs on a desktop GPU. Post-training INT8 quantisation using the TF Lite converter dropped the model to 3.8 MB and pushed inference to 10 FPS on the Pi, which was the threshold we needed for the system to feel responsive.

B. Sensor Anomaly Engine

The anomaly engine is deliberately simple. Each sensor channel gets its own independent rolling window and baseline. We tried a few fancier approaches — isolation forests, one-class SVMs — but found that the z-score threshold method worked just as well for the fault types

we cared about and was far easier to tune and explain. The two-tier alert system (soft at z = 2.5, hard at two consecutive breaches) came directly from observing the false alarm behaviour during motor spin-up in early bench tests. Single-sample threshold crossings during high vibration were constant and useless; requiring two in a row killed almost all of them.

C. Dashboard Integration

The Node-RED dashboard is the part of the system that the end user actually sees, so we spent more time on it than the code complexity might suggest. The rate limiter on the video topic (capped at 2 FPS) was necessary because uncapped image publishing was saturating the MQTT broker during multi-fault scenarios and causing telemetry messages to queue up. The SQLite log gives us a proper record of each mission that we can load into a spreadsheet or visualisation tool afterwards, which turned out to be very useful during the evaluation phase for checking whether the system had caught everything it should have.

VIII. EXPERIMENTAL RESULTS

We ran our validation tests in a 6 x 4 metre indoor enclosure that we set up to mimic a small section of industrial piping. The network used 25 mm galvanised steel pipe. For corrosion test specimens, we acid-etched pipe couplings until they showed visible surface degradation. Crack samples were cut with a 0.3 mm rotary burr. Gas release events were staged using a controlled nozzle delivering a nitrogen-propane mix. In total we ran 47 flight passes across three separate test sessions. That gave us a reasonably sized evaluation set without taking weeks.

A. Classification Accuracy

Table II shows the precision and recall numbers we got for each fault category.

TABLE II. PER-CATEGORY DETECTION METRICS

Category	Precision (%)	Recall (%)
Oxidation	89.3	88.7
Fracture	91.1	90.4
Nominal surface	95.2	96.0
Vapour release	90.6	89.9
Aggregate	91.6	91.2

Normal surfaces scored highest, which makes sense — they made up the bulk of the training data and have the most consistent appearance. Oxidation recall was the weakest of the fault categories. When we went back and looked at the missed cases, almost all of them came from frames where specular reflection off the pipe surface had blown out part of the image. We have since added a histogram equalisation step to the preprocessing pipeline that should help with this.

B. Latency

We measured end-to-end latency as the gap between frame capture and MQTT alert publication, using microsecond timestamps at each stage. Across 200 triggered fault events, median latency was 3.2 seconds and the 95th percentile was 4.7 seconds. Both are well inside our five-second target. For comparison, we also tested a version of the pipeline that forwarded frames to an external API for inference — that configuration averaged 15.4 seconds on the same Wi-Fi network. The difference makes a real argument for keeping inference onboard.

C. Sensor Performance

Gas detection caught 44 of 47 staged releases (93.6%). The three misses were all very low flow-rate releases, basically at the bottom of what the MQ-2 can reliably detect. Adding a more sensitive photo-ionisation sensor is on our list for the next hardware revision. Vibration anomalies from our shaker rig were caught perfectly above 50 Hz but only 80% of the time below that, which reflects the SW-420's known drop-off at lower frequencies. The ultrasonic sensor prevented all 15 simulated close-approach collisions without any issues. GPS accuracy during outdoor tests averaged 2.8 m CEP, which is good enough to point a maintenance team at the right pipe segment.

D. Comparison with Prior Work

Table III summarises how SurveilX compares to the existing approaches we reviewed.

TABLE III. CAPABILITY COMPARISON

Dimension	Prior Approaches	SurveilX
Spatial reach	Fixed nodes only	Repositionable UAV
Inference location	Remote / cloud	Fully onboard
Sensor diversity	1 or 2 types	5 independent types
Alert latency	10 – 30 s	3.2 s median
Detection accuracy	80 – 88 %	91.6 % aggregate
Operator risk	Direct exposure	Remote oversight

IX. CONCLUSION AND FUTURE SCOPE

We built SurveilX to answer a straightforward question: can you do industrial pipeline inspection better with a drone and onboard AI than with the methods people use today? Based on our results, the answer seems to be yes, at least for the fault types and environments we tested. A 91.6% aggregate accuracy and 3.2-second alert latency are numbers we are genuinely happy with, and the comparison to cloud-dependent alternatives shows why the edge-first architecture was the right call.

One thing we appreciated about the four-layer design was how much it simplified debugging and iteration. When the video latency was too high in early tests, we only had to look at the edge processing layer. When the MQTT broker got saturated, we fixed it in the communication layer. Nothing about changing the rate limiter touched the sensor logic or the dashboard. That kind of separation makes the system much easier to work on in a team of four people with overlapping responsibilities.

There is plenty still to do. The most obvious next step is autonomous navigation — right now someone has to fly the drone, which partially defeats the purpose. We want to get ROS 2 running with waypoint planning and LiDAR obstacle avoidance so the system can do a full inspection sweep without a human at the controls. Beyond that, running multiple drones in coordination would let you cover a large facility quickly, which is where the real scalability gain is. On the model side, we think moving to EfficientDet would help with the small-feature detection cases where MobileNetV2 struggles at altitude. And eventually, connecting the alert log to a cloud database for trend analysis would turn SurveilX from a real-time detector into something that can predict failures before they happen. That feels like the most valuable long-term direction.

REFERENCES

[1] S. Kumar and R. Sharma, "Industrial Fault Diagnosis Using Vibration and Temperature Sensors in IoT Framework," IEEE Sensors Journal, vol. 22, no. 4, pp. 3456-3465, Feb. 2022.
[2] L. Zhang, H. Wang, and Y. Liu, "UAV-based Pipeline Monitoring System Using IoT and Deep Learning," IEEE Access, vol. 9, pp. 112034-112045, 2021.
[3] M. Ali, F. Khan, and S. Ahmed, "Real-Time Leak Detection in Pipelines using Machine Learning and IoT," IEEE Internet of Things Journal, vol. 7, no. 10, pp. 9552-9561, Oct. 2020.
[4] RandomNerdTutorials, "ESP32 with DHT22 Sensor," [Online]. Available: https://randomnerdtutorials.com/esp32-dht11-dht22-temperature-humidity-sensor-arduino-ide/
[5] SunFounder, "ESP32 MQ-2 Gas Sensor Tutorial," [Online]. Available: https://docs.sunfounder.com/projects/umsk/en/latest/03_esp32/esp32_lesson04_mq2.html
[6] SunFounder, "ESP32 SW-420 Vibration Sensor Tutorial," [Online]. Available: https://docs.sunfounder.com/projects/umsk/en/latest/03_esp32/esp32_lesson24_vibration_sensor.html
[7] RandomNerdTutorials, "ESP32 HC-SR04 Ultrasonic Sensor," [Online]. Available: https://randomnerdtutorials.com/esp32-hc-sr04-ultrasonic-arduino/
[8] RandomNerdTutorials, "ESP32 NEO-6M GPS Module," [Online]. Available: https://randomnerdtutorials.com/esp32-neo-6m-gps-module-arduino/
[9] A. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv preprint arXiv:1704.04861, 2017.
[10] Eclipse Foundation, "Eclipse Mosquitto: An Open Source MQTT Message Broker," [Online]. Available: https://mosquitto.org/