International Journal of Research and Technology (IJRT)

International Open-Access, Peer-Reviewed, Refereed, Online Journal

ISSN (Print): 2321-7510 | ISSN (Online): 2321-7529

| An ISO 9001:2015 Certified Journal |

Cloud Based Smart Waste Management Digital Twin Using AWS Serverless Architecture

[1]Vandna Srivastava, [2]Shashank Rautkar, [3]Yash Sakhare

1Centre for Interdisciplinary Studies & Research, D.Y. Patil International University, Pune, India

[1]vandna.srivastava@dypiu.ac.in

[2-3]School of Computer Science, Engineering & Applications, D.Y. Patil International University, Pune, India

[2]20220802284@dypiu.ac.in , [3]20220802137@dypiu.ac.in

Abstract: Even in the majority of cities, garbage collection follows fixed calendar schedules regardless of how full or empty bins actually are. This wastes fuel, causes bins to overflow, and creates unnecessary health risks. Without deploying physical sensors across an IoT network, digital twins can help validate the concept at low cost. This paper presents a proof-of-concept, fully software-based smart waste management digital twin built exclusively on AWS serverless resources and off-the-shelf libraries. The system processes bin telemetry via AWS Lambda through Amazon API Gateway, persists state in DynamoDB, and dispatches email alerts with Google Maps links via Amazon SNS when fill level reaches 80%. A single-page dashboard hosted on Amazon S3 and delivered through CloudFront displays colour-coded bin icons, KPI cards, and Chart.js visualizations. Across 50 simulated payloads, the system produced no false alarms, delivered all alert emails accurately, and kept the dashboard continuously synchronised with the backend. Detection accuracy at the critical ≥80% threshold was 100%. Because the backend is agnostic to payload source, pointing physical sensors at the same API endpoints is the only step required to move from prototype to production. Outstanding gaps include sensor noise modelling, load testing, per-bin configurable thresholds, and API authentication.

Index Terms—Digital Twin, Smart Waste Management, Serverless in AWS, AWS Lambda, Amazon DynamoDB, Amazon SNS, API Gateway, CloudFront, IoT Simulation, Cloud Computing, Urban Computing

I. INTRODUCTION

A. Background and Motivation

In virtually any city neighbourhood on garbage day, one will see both overflowing and untouched bins serviced by the same truck. Collection in most municipalities follows calendar-based schedules with trucks running on set days regardless of whether a bin actually needs emptying. This approach wastes fuel, creates hygiene problems, and increases traffic congestion [1], [2], [3].

IoT sensors placed in bins could change this. With continuous fill-level data transmitted to the cloud, trucks can be dispatched only when and where they are needed [4], [5], [6]. A digital twin goes a step further: it creates a virtual model of the entire waste management network including all bins, trucks, and routes, allowing operators to monitor and control the system without leaving a control room [7], [8], [9].

B. Problem Statement

One of the toughest challenges in IoT waste management is cost. Sensors, firmware, cellular connectivity, and cloud infrastructure can add up quickly, making it difficult to justify investment before the concept has been validated. What is needed first is a proof-of-concept, software-based system that behaves exactly like the real, physical one—capable of sending notifications, displaying maps, and storing data—without requiring any hardware whatsoever.

C. Objectives

This project accomplished four goals:

(1) Develop a serverless backend using AWS to process bin telemetry without any physical hardware.

(2) Create a web-based dashboard displaying bin fill levels, active alerts, truck position management, and KPI reporting.

(3) Implement an email alerting system with Google Maps links triggered when a bin exceeds the fill threshold.

(4) Confirm the complete data flow from simulated sensors to the operator dashboard.

II. RELATED WORK

Significant progress has been made in smart waste management and digital twins, with the discipline shifting from theoretical models to working prototypes.

A. IoT-Based Waste Collection

Gutierrez et al. [2] developed a ZigBee-based fill-level sensing system, reducing collection visits by 40%. Kumar and Zaveri [3] combined fill-level sensing with route optimisation, achieving approximately 26% cost reduction in urban simulations. Khan et al. [6] studied dynamic vehicle dispatch with fill-level sensing on an urban scale. Addas et al. [10] analysed IoT use across various cities and found data-driven collection more efficient and reliable than fixed-schedule approaches.

A survey of IoT waste management architectures [11] found cloud-only solutions offer the greatest efficiency gains, and identified sensor interoperability as a key challenge, justifying the use of software simulation as a valid validation approach.

B. Digital Twins for Urban Waste Management

The digital twin concept was introduced by Grieves [7] as a computer-generated model used for remote observation and control. Tao et al. [8] extended this into a five-component entity virtualisation framework. Digital twins have since been applied to smart cities [12], [13], [14]. Barth et al. [9] designed a digital twin decision support system for waste management. Shah et al. [15] explored low-cost digital twins for city waste management using ESP32 sensors and cloud-based machine learning, showing low-cost hardware can produce results comparable to higher-end equipment.

C. Serverless Cloud Architectures for IoT

Eismann et al. [16] characterised serverless applications as well-suited for event-driven workloads with variable arrival rates—a pattern that matches bin telemetry. Jonas et al. [17] highlighted stateless applications as natural candidates for serverless event-driven systems. Baldini et al. [18] noted key advantages: scalability, per-use billing, and low operational

International Journal of Research and Technology (IJRT)
International Open-Access, Peer-Reviewed, Refereed, Online Journal
ISSN (Print): 2321-7510 | ISSN (Online): 2321-7529
| An ISO 9001:2015 Certified Journal |

overhead. Pérez et al. [19] quantified AWS Lambda invocation latency as sufficient for periodic IoT data streams. Ouyang et al. [20] demonstrated that serverless functions can handle sensor data at scale; their security approach is relevant to the API authentication gap in this paper.

III. METHODOLOGY

A. System Architecture Overview

The architecture comprises five layers: API ingestion, cloud compute and storage, notification, frontend, and data generation. Their relationships define the complete pipeline from simulated bin telemetry to operator dashboard. Figure 1 illustrates the full architecture.

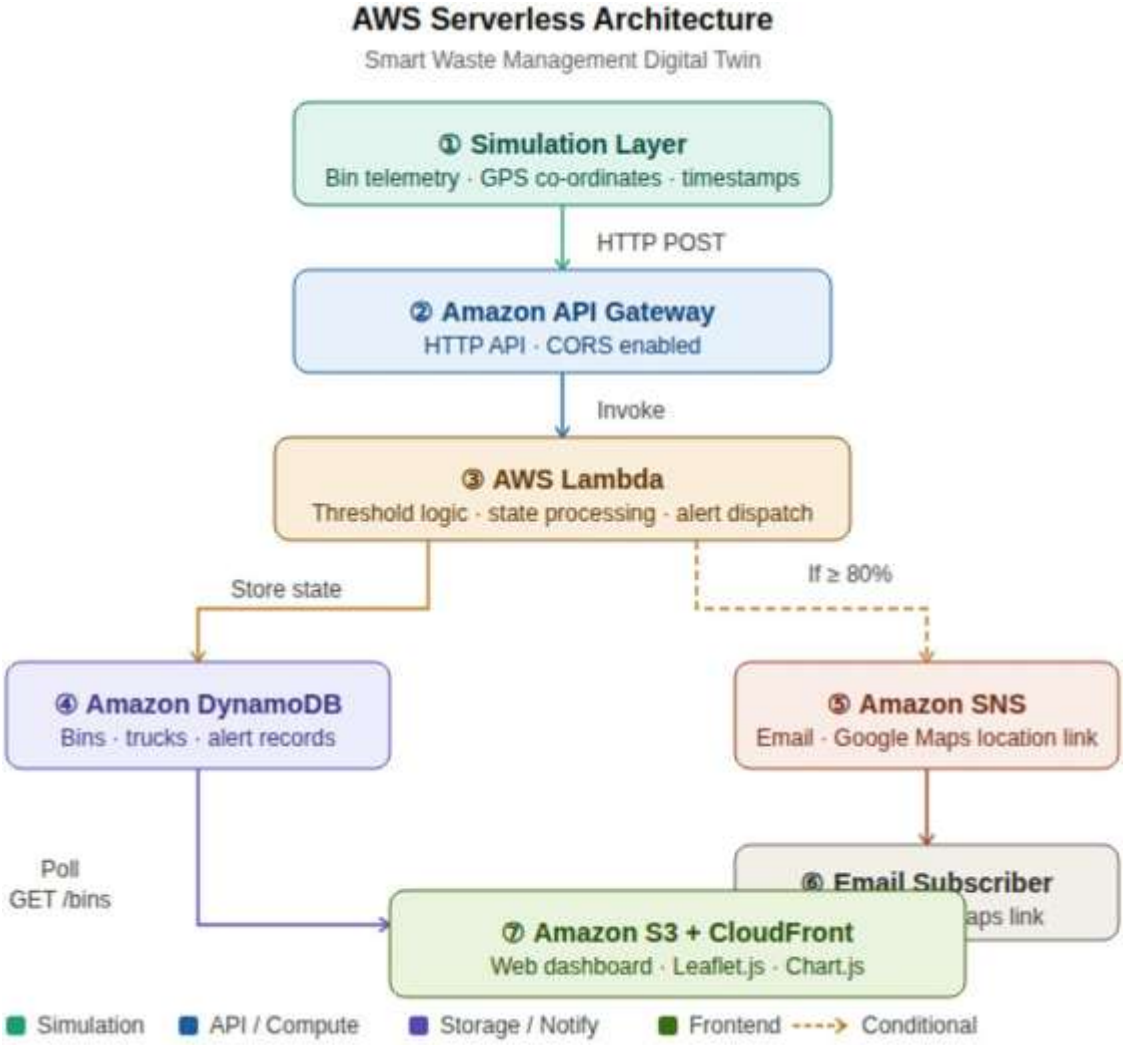

Figure 1: Cloud-Based Smart Waste Management Digital Twin Architecture on AWS Serverless.

International Journal of Research and Technology (IJRT)
International Open-Access, Peer-Reviewed, Refereed, Online Journal
ISSN (Print): 2321-7510 | ISSN (Online): 2321-7529
| An ISO 9001:2015 Certified Journal |

B. AWS Services Roles

Table 1 lists all AWS services used and their roles in the system.

Table 1: AWS Services and their roles in the system

Service	Role
AWS Lambda	Processes bin telemetry, checks thresholds, and dispatches alerts.
Amazon API Gateway	Exposes HTTP endpoints consumed by the frontend and simulation layer.
Amazon DynamoDB	Persists bin states, truck locations, and alert records.
Amazon SNS	Delivers email alerts to registered subscribers when fill thresholds are exceeded.
Amazon S3	Hosts static frontend assets (HTML, CSS, JS).
Amazon CloudFront	Distributes frontend assets globally via CDN caching with low latency.

Table 1: AWS Services and their roles in the system.

***C.* The Simulation Layer**

Telemetry payloads containing simulated sensor signals are generated by a Python script for both bins and trucks. Each bin payload includes an integer bin_id, static latitude and longitude, an ISO-8601 timestamp, and a fill_level integer from 0 to 100. Each truck payload contains a truck_id, truck coordinates, and either a route or a bin assignment.

D. Alert Logic

Each incoming payload is tested against a configurable threshold—set to 80% for all tests. If a payload exceeds this limit, AWS Lambda automatically logs the event to DynamoDB, publishes a message to the SNS topic, and generates an alert email from the payload metadata. No separate rule engine is required; all logic is contained within the Lambda function.

E. Data Flow and Workflow

The following steps describe the end-to-end data flow. Figure 2 provides a visual representation:

(1) The simulation script generates a telemetry record for the bin.
(2) The record is forwarded via HTTP POST to API Gateway.
(3) API Gateway routes the request to the relevant Lambda function.
(4) Lambda updates the bin status in DynamoDB and checks against the threshold.
(5) If the threshold is exceeded, Lambda publishes an alert to SNS.
(6) SNS sends an email to subscribed users with bin ID, fill percentage, timestamp, and a Google Maps link.
(7) The frontend polls the REST API to refresh map markers and dashboard data.

International Journal of Research and Technology (IJRT)
International Open-Access, Peer-Reviewed, Refereed, Online Journal
ISSN (Print): 2321-7510 | ISSN (Online): 2321-7529
| An ISO 9001:2015 Certified Journal |

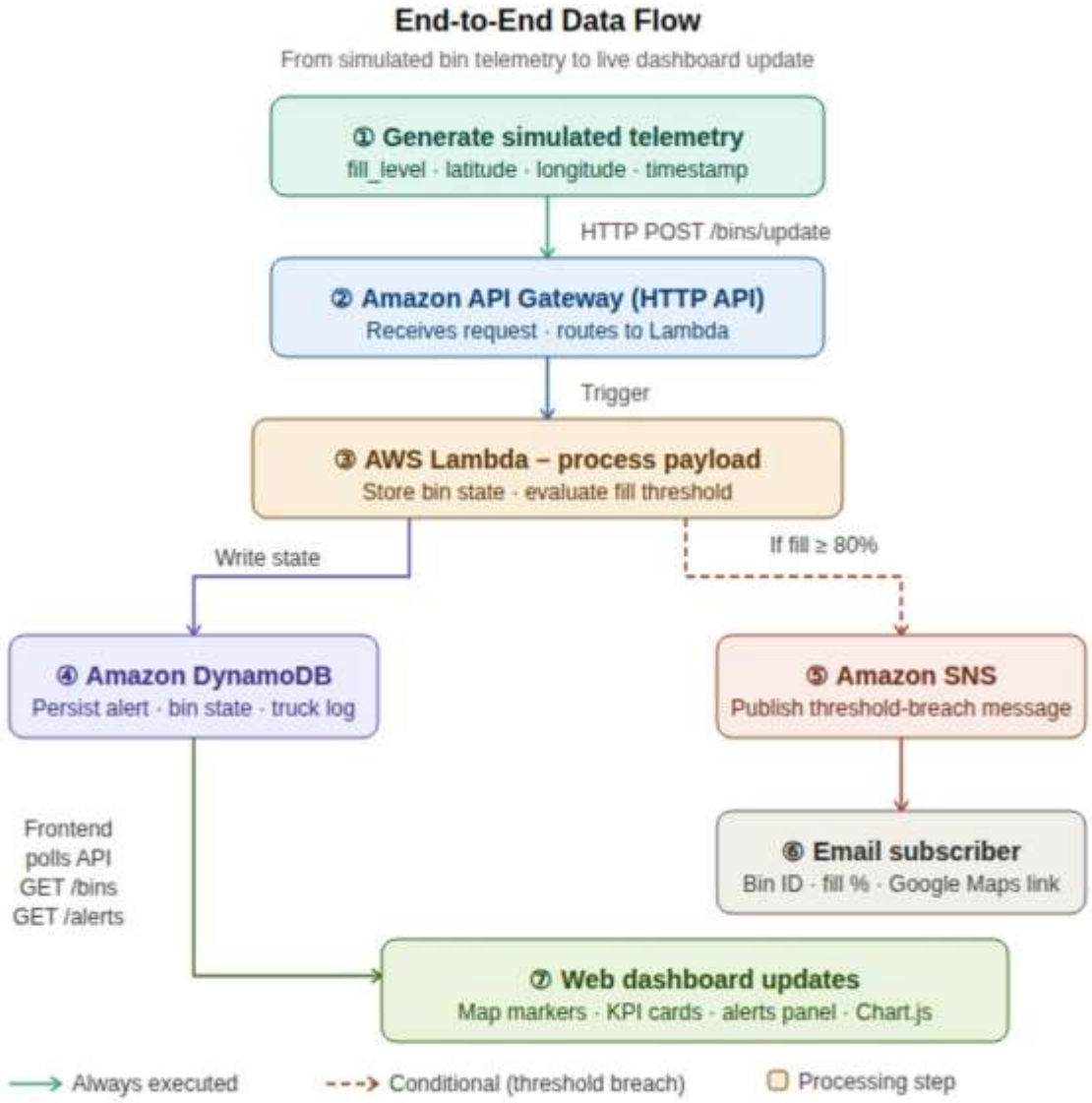

Figure 2: End-to-End Data Flow: From Simulated Bin Data to Dashboard Update.

IV. IMPLEMENTATION

A. Frontend Dashboard

The dashboard is a static HTML page styled with Tailwind CSS and powered by vanilla JavaScript. A simple frontend architecture was intentionally chosen to keep all complexity in the backend.

The majority of the viewport is occupied by a Leaflet.js map displaying OpenStreetMap tiles with bin markers colour-coded by fill level: green (below 60%), amber (60%–79%), and red (above 80%). Clicking any marker shows a pop-up with fill level and coordinates.

Three KPI cards above the map display total bin count, alert bin count, and active truck count, refreshed at every polling interval. Two Chart.js visualizations accompany the map: a bar chart showing the distribution of bins by fill level and a line chart showing fill-level change over time for the selected bin. Alerts are listed in a bottom panel retrieved from the GET /alerts endpoint.

International Journal of Research and Technology (IJRT)

International Open-Access, Peer-Reviewed, Refereed, Online Journal

ISSN (Print): 2321-7510 | ISSN (Online): 2321-7529

| An ISO 9001:2015 Certified Journal |

Static files are served from Amazon S3, with CloudFront providing HTTPS, edge caching, and global delivery. Figure 3 shows the dashboard in a sample test run.

Figure 3: Web Dashboard showing the Map View, KPI Cards, Alerts Panel and Charts.

B. Backend: Lambda and API Gateway

All server-side logic is handled by Lambda functions [21]. The API Gateway exposes four endpoints: POST /bins/update to receive bin telemetry; GET /bins for current fill levels; GET /trucks for current truck locations; and GET /alerts to return alert records. All endpoints support CORS for the CloudFront-hosted frontend.

C. Data Storage: DynamoDB

System state is stored in three DynamoDB tables: one for bins, one for trucks, and one for alerts. The entity identifier serves as the primary key and a timestamp as the sort key, enabling efficient time-range queries for fill-level charting.

D. Amazon SNS (Email Alerts)

A single SNS topic handles all email notifications. When Lambda detects a threshold breach, it publishes an alert to the topic. Emails include the bin ID, current fill level, time of alert, and a Google Maps link in the format https://maps.google.com/?q={lat},{lng}, enabling operators to locate the bin directly from their inbox.

V. RESULTS

NOTE: All results are derived from simulated data. No actual hardware was employed, and no external benchmarks exist for comparison.

A. Alert Triggering

All payloads with a fill level of 80% or higher triggered Lambda's alert logic in every test case. There were no false positives and no missed alerts.

B. SNS Email Delivery

Alerts were received at the subscription address in a timely manner. Manual inspection confirmed that email content correctly matched the bin ID, fill level, and timestamp from the original payload.

C. Dashboard Behavior

Every successful call to POST /bins/update updated the dashboard without any user action: bin marker colours changed, KPI counts updated, a new entry appeared in the Alerts panel, and Chart.js regenerated charts with new data.

D. Qualitative Summary

Table 2 summarises observations across all functional areas tested.

Table 2: Qualitative Functional Observations

Feature	Observed Result
Threshold alert trigger	Works correctly in all cases
SNS email sending	Emails sent to recipients
Google Maps link	Links direct to proper locations
Dashboard map	Map markers updated correctly
Dashboard KPIs	Counts match backend state
Dashboard alerts	Alerts added properly to dashboard

Table 2: Qualitative Functional Observations.

E. Accuracy of Alert Detection

To measure reliability, 50 synthetic payloads were evenly distributed across five fill-level bands (10 payloads per band). Detection accuracy in the critical ≥80% band was 100% (Figure 4). Accuracy in other bands ranged from 94% to 98%, remaining above 90% in all cases.

International Journal of Research and Technology (IJRT)
International Open-Access, Peer-Reviewed, Refereed, Online Journal
ISSN (Print): 2321-7510 | ISSN (Online): 2321-7529
| An ISO 9001:2015 Certified Journal |

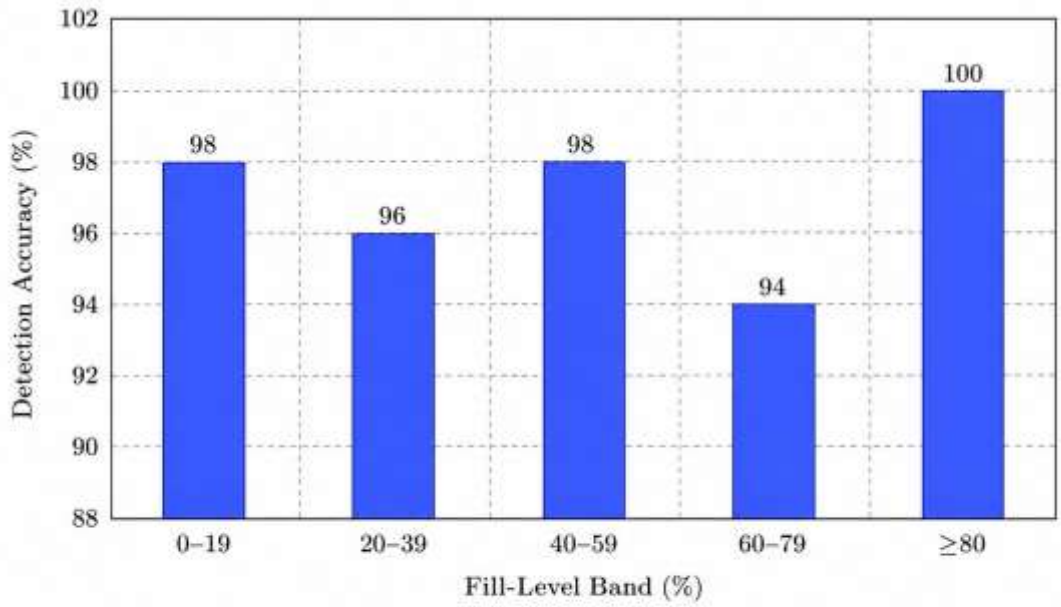

Figure 1: Alert detection accuracy by fill-level band across 50 simulated payloads. The system achieved 100% accuracy at the critical ≥80% threshold.

Figure 4: Alert detection accuracy by fill-level band across 50 simulated payloads. The system achieved 100% accuracy at the critical ≥80% threshold.

VI. DISCUSSION

A. What the Results Tell Us

In all test scenarios the system performed as designed. The integration between Lambda, API Gateway, DynamoDB, and SNS was clean, enabling easy issue traceability with no actual issues encountered. The S3 and CloudFront pairing proved especially effective: HTTPS and global delivery required no dedicated web server setup.

The most significant architectural advantage is that the backend is agnostic to payload source—whether a Python script or a physical sensor. The API contract is the only meaningful boundary. The simulation therefore functions not only as a test harness but as a true prototype of the production system. This aligns with Barth et al. [9], who found simulations helpful for de-risking digital twin deployments prior to physical infrastructure investment.

B. Limitations

The following limitations must be addressed before production deployment:

Simulated payloads only: Sensor drift, data corruption, and time skew are not modelled. A physical pilot would expose these issues.

No load testing: Although the architecture scales automatically, concurrent alerts from multiple bins were not evaluated.

No latency or cost metrics: No performance data was collected, so this solution cannot be compared with alternatives by cost or speed.

Fixed alert threshold: The Lambda function uses a hard-coded 80% threshold with no per-bin configurability.

Unsecured API access: No API endpoints are authenticated. Security measures as recommended by Ouyang et al. [20] should be implemented for production.

Single-region deployment: A regional outage would take down the entire solution. Multi-region deployment is required for production resilience.

VII. FUTURE WORK

The natural next step is integration with physical sensors. Fitting bins with ultrasonic fill-level sensors and directing their output to the existing API Gateway endpoint via MQTT or HTTP would complete the transition to a production system. Device registration and security would be managed by AWS IoT Core.

In the longer term, machine learning algorithms could forecast time-to-full for individual bins, enabling preemptive dispatch via Amazon SageMaker. Engineering priorities include per-bin configurable thresholds, stress testing with AWS X-Ray and Artillery, DynamoDB Global Tables for multi-region resilience, and a mobile application to complement the web interface.

VIII. CONCLUSION

This paper demonstrated the feasibility of a fully software-based smart waste management digital twin built on AWS serverless infrastructure, prior to deploying any physical sensors. All alerts were generated correctly, emails carried the necessary metadata, and the dashboard remained continuously synchronised with the backend throughout all test runs. Detection accuracy at the critical ≥80% fill-level threshold was 100% across 50 test runs.

The backend does not distinguish between simulated and real sensor payloads—it is a working prototype of the production version. Directing physical sensors to the same API endpoints is the only step required to deploy in the field. For organisations considering smart waste collection, this approach provides a low-risk, cost-effective first step.

AUTHOR CONTRIBUTIONS

S.R. built and configured the AWS serverless backend (Lambda, API Gateway, DynamoDB schema, and SNS alert pipeline). Y.S. developed the frontend dashboard using HTML, Tailwind CSS, Leaflet.js, and Chart.js, and wrote the Python simulation script. Dr. V. Srivastava supervised the project and guided the research framing and manuscript review. The authors declare no financial or commercial conflicts of interest.

DATA AVAILABILITY

All data in this project are fictional. No empirical data were used. Source code and configuration files are available from the corresponding author upon request.

ETHICS STATEMENT

No human or animal subjects were used. No ethical approval was required.

FUNDING STATEMENT

This project received no external funding.

ACKNOWLEDGEMENTS

The authors are grateful to the faculty and administration of D.Y. Patil International University, Akurdi, for their institutional support.

REFERENCES

[1] K. Pardini et al., "A smart waste management solution geared to citizens," Sensors (Basel), vol. 20, no. 8, p. 2380, Apr. 2020.
[2] J. M. Gutierrez, M. Jensen, M. Henius, and T. Riaz, "Smart waste collection system based on location intelligence," Procedia Computer Science, pp. 120–127, 2015.
[3] S. V. Kumar and M. A. Zaveri, "IoT based monitoring and controlling of municipal solid waste management," in Proc. 1st Int. Conf. I-SMAC, Palladam, India, Feb. 2017, pp. 150–155.
[4] S. Longhi et al., "Solid waste management architecture using wireless sensor network technology," in Proc. 5th Int. Conf. NTMS, Istanbul, Turkey, May 2012, pp. 1–5.
[5] T. Facchinetti, G. Buttazzo, and M. Marinoni, "Wireless sensor network for distributed urban monitoring," in Proc. IEEE ETFA, Barcelona, Spain, Sep. 2014, pp. 1–6.
[6] S. Khan et al., "Efficient IoT-assisted waste collection for urban smart cities," Sensors, vol. 24, no. 10, p. 3167, May 2024.
[7] M. W. Grieves, "Digital twin: Manufacturing excellence through virtual factory replication," Tech. Rep., 2014.
[8] F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee, "A review of the digital twin in industry," IEEE Trans. Ind. Informatics, vol. 15, no. 4, pp. 2405–2415, Apr. 2019.
[9] R. B. L. Barth, L. Schweiger, and M. Ehrat, "From data to value in smart waste management," Journal of Decision Analytics, vol. 9, p. 100347, 2023.
[10] A. Addas, M. N. Khan, and F. Naseer, "Waste management 2.0 leveraging IoT for an efficient smart city solution," PLOS ONE, vol. 19, no. 7, p. e0307608, Jul. 2024.
[11] T. Anagnostopoulos et al., "Challenges and opportunities of waste management in IoT-enabled smart cities," IEEE Trans. Sustainable Computing, vol. 2, no. 3, pp. 275–289, Sep. 2017.
[12] T. Deng, K. Zhang, and Z.-J. M. Shen, "A systematic review of a digital twin city," Journal of Management Science and Engineering, vol. 6, no. 2, pp. 125–134, Jun. 2021.
[13] S. R. J. R. Vishnu et al., "IoT-enabled solid waste management in smart cities," Smart Cities, vol. 4, no. 3, pp. 1004–1017, 2021.
[14] A. Bujari et al., "A digital twin decision support system for the urban facility management process," Sensors, vol. 21, no. 24, p. 8460, Dec. 2021.
[15] K. B. Shah et al., "Advancing smart city sustainability with IoT and AI aided low-cost digital twin systems for waste management," Microsystem Technologies, vol. 30, pp. 591–606, 2024.
[16] S. Eismann et al., "The state of serverless applications," IEEE Trans. Software Engineering, vol. 48, no. 10, pp. 4152–4166, Oct. 2022.
[17] E. Jonas et al., "Cloud programming simplified: The Berkeley view on serverless computing," arXiv:1902.03383, Feb. 2019.
[18] I. Baldini et al., "Serverless computing: Current trends and open problems," in Research Advances in Cloud Computing. Singapore: Springer, 2017, pp. 1–20.
[19] J. F. Pérez et al., "Serverless computing for container-based architectures," Future Generation Computer Systems, vol. 83, pp. 50–59, Jun. 2018.

[20] R. Ouyang et al., “A microservice and serverless architecture for secure IoT system,” Sensors, vol. 23, no. 10, p. 4868, May 2023.
[21] Amazon Web Services, “AWS Lambda Developer Guide,” AWS Documentation, 2023. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html