Journal of Theoretical and Applied Information Technology
15th November 2025. Vol.103. No.21

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

OPTIMIZED SERVERLESS FRAMEWORK FOR SCALABLE DEPLOYMENT OF DEEPAR+ FORECASTING

GANESHAN MAHALINGAM[1], RAJESH APPUSAMY[2]

[1]Cloud SME, Jain (Deemed to be University), Department of Computer Science, Kanakapura, India

[2]Professor. Jain (Deemed to be University), Department of Computer Science, Kanakapura, India

E-mail: [1]link2ganeshan@gmail.com, [2]a.rajesh@jainuniversity.ac.in

ABSTRACT

This study proposes an optimized serverless framework for the scalable and cost-efficient deployment of DeepAR+ forecasting models, addressing critical challenges in time-series inference such as cost overhead, latency and operational complexity. The architecture integrates AWS SageMaker Batch Transform and OpenWhisk containerized actions to enable on-demand, low-cost model execution, achieving up to 40% reduction in operational expenses compared to always-on endpoints. To minimize latency, precomputed DeepAR+ predictions are cached in Redis, and Apache Kafka decouples heavy inference workloads from real-time processing, allowing sub-second anomaly detection. Security is strengthened through VPC-isolated deployments, IAM-based access control and real-time auditing via CloudTrail, improving unauthorized access detection to 99.9%. A lightweight pre-processing pipeline ensures consistent input data, even in sparse telemetry environments. The proposed solution delivers a secure, scalable and efficient forecasting framework that leverages the flexibility of serverless infrastructure while maintaining real-time responsiveness, making it well-suited for modern predictive analytics applications in data-intensive environments.

Keywords: *DeepAR+, Serverless, SageMaker, AWS IoT Core, AWS Glue, OpenWhisk*

1. INTRODUCTION

The rapid evolution of Intelligent System such as cloud based IoT has spurred the need for scalable, real-time predictive capabilities in various sectors such as smart cities, healthcare, logistics and industrial automation. Predictive analytics becomes central to facilitating proactive decision-making, anomaly identification and resource management when the amount and speed of telemetry information grow. This study is necessary to address the growing demand for scalable, secure, and real-time predictive analytics in cloud-IoT systems. Existing DeepAR+ deployments are hampered by high latency, high cost, and poor security. Current approaches optimize one factor at a time without considering an integrated solution. By combining AWS SageMaker Batch Transform, OpenWhisk actions, Redis caching, Kafka decoupling, AWS Glue preprocessing, and IAM-based security, this work provides an end-to-end, cost-effective, and production-grade serverless architecture for DeepAR+ forecasting. The architecture is based on cloud-native and event-driven services and incorporates AWS SageMaker Batch Transform for low-cost inference, Apache OpenWhisk containers for efficient execution, Redis caching, and Apache Kafka for decoupled processing with low latency. Security is enhanced via VPC-isolated model endpoints, IAM-based access control, and CloudTrail auditing, while AWS Glue enables automated pre-processing of sparse and irregular telemetry data. By large-scale evaluation, the suggested framework achieves considerable improvements in cost savings (up to 40%), sub-second response time for anomaly detection and strong security enforcement. This study introduces a practical and scalable approach for deploying intelligent forecasting models with serverless design patterns, providing improved efficiency for today's data-intensive environments.

2. RELATED WORK

Deep Learning Models for IoT Time-Series Prediction - Recent research has used deep learning models including LSTM, GRU, TCN,

and DeepAR+ to make predictions in IoT applications for forecasting sensor readings and anomaly detection. DeepAR+, a product by Amazon, has become an increasingly popular tool because it can return probabilistic forecasts for multiple time series. Although DeepAR+ is offering encouraging outcomes in smart energy, traffic forecasting, and industrial monitoring applications, most use cases are dependent on managed services such as Amazon SageMaker, which may be expensive and rigid in large-scale, real-time usage [1][2].

Deployment Strategies for DeepAR+ on AWS - DeepAR+'s standard deployment options are either real-time endpoints (SageMaker-hosted models) or batch transform jobs. Research reports that despite real-time endpoints providing low-latency predictions, they come with high operation expenses and poor scalability in bursty IoT settings [3]. In contrast, batch transforms are operationally cheap but unsuitable for real-time application. Existing work does not address serverless, containerized deployment patterns—especially those that disentangle inference from actuation to optimize for performance

Limitations of Serverless Platforms (AWS Lambda vs. OpenWhisk) Although highly popular, AWS Lambda has documented limitations such as cold start latency, limited runtime environments, and lack of customization. It has been studied in the past comparing Lambda to Apache OpenWhisk, pointing out OpenWhisk's flexibility in handling custom Docker runtimes and its applicability for Kubernetes-based deployments [4]. Few studies, however, have compared OpenWhisk's integration with complex ML workloads like DeepAR+ in Cloud IoT environments particularly in the context of cost, performance, and interoperability.

IoT-cloud integration security is a persistent issue, with the key areas of focus being identity management, encryption, secure model access, and auditing. Research has highlighted IAM-based access control, VPC isolation for ML endpoints, and audit logging via services such as AWS CloudTrail [5]. Yet few current models provide a single security framework that integrates these aspects with serverless computing and machine learning in Cloud IoT-specific implementations.

2.1 Gaps in Existing Approaches

Overall, prior research often addresses forecasting accuracy or cost individually, but few solutions provide a holistic, secure, and cost-optimized framework for DeepAR+ deployment in IoT architectures. Little attention has been paid to:

Decoupling inference from device actuation pipelines, Leveraging caching for latency mitigation, orchestrating batch and containerized models using serverless platforms like OpenWhisk. This study aims to address these gaps by designing and evaluating a modular, secure, and serverless framework for DeepAR+ in scalable IoT scenarios.

2.2 Study Gaps and Future Directions

Gaps in This Work:

1) Framework is AWS-dependent, limiting portability across other clouds or on-premises systems.

2) DeepAR+ used as a single model without ensemble benchmarking.

Future Directions:

1) Extend to Azure, Google Cloud, and Knative/ OpenWhisk for cross-cloud portability.

2) Enable multi-model ensembles (DeepAR+, Prophet, LSTM) via SageMaker or OpenWhisk for optimal accuracy–latency trade-offs. Single-Model Focus (DeepAR+ Only) - The study evaluates only DeepAR+ forecasting. Other models such as LSTM, Prophet, or ARIMA were not tested, and no ensemble techniques were implemented.

3. PROBLEM STATEMENTS AND OBJECTIVES

3.1 High Operational Cost of Real-Time DeepAR+ Inference in Large-Scale Cloud IoT Systems

Using persistent endpoints to deploy DeepAR+ to perform inference in real-time on AWS SageMaker is highly operationally costly because ongoing provisioning of compute resources is required. This legacy approach to deployment is not efficient in scenarios involving massive intelligent systems and Cloud IoT mashups, where prediction is only usually needed intermittently or upon the occurrence of an event. Consequently, massive compute resources go idle, causing avoidable spending and wastage of infrastructure.

3.2 Latency Constraints in Real-Time Anomaly Detection Using DeepAR+

Although it performs high-accuracy predictions, it is plagued by perceivable inference latency under heavy device workloads or high-rate data streams. In applications with sensitive latency, like anomaly detection in safety-critical systems, this latency hinders real-time response and detracts from the value of predictive analytics in serverless environments.

3.3 Inadequate Security controls in Serverless Integration of DeepAR+ Forecasting

Existing cloud-native stacks combining serverless computing (Apache OpenWhisk) with predictive models do not typically support strong, granular security measures. Poor encryption, insecure model endpoint access, and limited auditing leave the system vulnerable to breaches, unauthorized access, and failure to meet data governance policies of particular concern in high-risk or regulated settings.

3.4 Research Objectives

1) Reduce Operational Costs of DeepAR+ Inference in Cloud IoT Deployments.

2) Minimize Inference Latency for Real-Time Anomaly Detection.

3) Enhance Security for Serverless DeepAR+ Integration in Cloud-Based IoT Systems

4) Ensure Data Consistency and Compatibility for DeepAR+ in Heterogeneous Cloud IoT.

This study emphasizes real-world practical trade-offs in IoT deployments, demonstrating how combining SageMaker Batch Transform with OpenWhisk containers decouples inference from actuation to transform architectures from event-latency provisioning to event-triggered execution, making enterprise-grade predictive analytics cheaper through decoupled inference and actuation. Three obvious conclusions result from our experiments:

1) Hybrid inference (SageMaker + OpenWhisk) achieves both cost efficiency and low-latency forecasting.

2) Redis caching converts DeepAR+ probabilistic outputs into sub-second responses vital for real-time anomaly detection.

3) Built-in security and observability (VPC, IAM) enhance compliance and access protection across the IoT pipeline.

4. PROPOSED FRAMEWORK ARCHITECTURE

4.1 IoT Devices → AWS IoT Core

In the Optimized Serverless Architecture for DeepAR+ shown in Figure 1, the flow starts with IoT devices sending real-time telemetry data to AWS IoT Core via lightweight protocols like MQTT or HTTP. This step is the basis of the overall architecture, as AWS IoT Core serves as the central ingestions layer that can manage millions of simultaneous device connections [7]. It gives a secure, scalable, and managed IoT data gateway to ensure that only legitimate and authorized devices are permitted to send data via IoT policies and mutual TLS authentication.

This ingestion mechanism directly facilitates the Objective (1) of the study lowering operational expenses by alleviating the task of secure data transport and device authentication to a managed service, thus avoiding the necessity to manually provision and manage individual custom ingestion endpoints. Managed by nature, AWS IoT Core can scale elastically according to workload without incurring idle infrastructure expenses [7], which is consistent with the cost-optimization objectives of serverless architectures.

At the same time, this aspect also aims at Objective (3) promoting security via the utilization of fine-grained AWS IoT policies. These policies regulate which devices are allowed to publish or subscribe to what topics, keeping the communication secure and isolated. Additionally, AWS IoT Core provides integration with IAM roles so that tightly scoped access permissions can be granted for downstream services like Lambda, OpenWhisk (using IAM-bound API Gateway), or message queues. Therefore, the information finds its way into the system in a controlled, secure, and cost-effective manner, providing the foundation for all downstream predictive analytics and decision-making in the proposed Architecture shown in Figure 1 .

4.2 Kafka/SQS → Glue-based ETL

In the SCOFI-IoT architecture, AWS IoT Core employs the Rules Engine to forward telemetry data [7] to Kafka which supports asynchronous and decoupled processing of data [9] [10]. This configuration isolates real-time ingestion from downstream workloads, which is essential for burst traffic handling without performance loss. This routing helps achieve Objective (2) by having the latency spikes minimized—as queued data can be separately

processed by inference components such as OpenWhisk and DeepAR+. It also serves Objective (4) by providing organized, time-sequence inputs to the ETL pipeline so that DeepAR+ has uniform and preprocessed data even from diverse IoT sources. The integration of Kafka's streaming at high throughput and SQS's durable queuing provides scalable, fault-tolerant, and latency-insensitive data transfer in the serverless architecture.

4.3 Glue → DeepAR+ Inference (SageMaker Batch)

In the (SAODF) architecture (Figure1), AWS Glue preprocessed data is passed on to the DeepAR+ inference engine, which is installed in a dual-mode setup utilizing either SageMaker Batch Transform for large-scale and cost-effective forecasting or containerized execution of DeepAR+ using OpenWhisk for real-time, on-demand forecasting. This deployment model with elasticity dramatically lowers operational expenditures by avoiding the overhead of having to run SageMaker endpoints continuously, thus resolving Objective (1). Concurrently, it provides low-latency responses by making local inference paths possible through OpenWhisk, facilitating Objective (2). This framework enables dynamic scaling of intelligent DeepAR+ forecasting according to Cloud IoT workload demand, striking a balance between performance and cost.

4.4 DeepAR+ Output → Redis Cache

In the SAODF Framework (1), the output from the DeepAR+ inference engine is stored in a Redis cache to enable rapid access by downstream components such as decision logic modules. This caching mechanism plays a critical role in minimizing redundant computations by serving previously generated predictions directly from memory, rather than triggering new inference runs. As a result, this significantly reduces latency and computational overhead, making the system highly responsive to recurring data patterns. By enabling near-instantaneous access to forecast results, the Redis layer supports Objective (2) facilitating sub-second anomaly detection in real-time.

4.5 Redis / Kafka →AWS EKS (OpenWhisk)

In the (SAODF) Framework (Figure 1), both Redis and Kafka serve as critical data feeders to the OpenWhisk action layer, which acts as the system's decision-making and orchestration hub. Redis delivers cached DeepAR+ predictions instantly, while Kafka streams real-time telemetry or event data [9]. The OpenWhisk functions, running in secure, containerized environments on EKS, fuse these inputs to execute context-aware decision workflows such as triggering alerts, updating dashboards, or issuing control signals to IoT devices. This decoupled,event-driven execution model supports massive scalability without the overhead of maintaining long-running services.Moreover, OpenWhisk enables fine-grained control over execution policies and integrates tightly with IAM roles for secure access, fulfilling Objective (2) by enabling low-latency responses and Objective (3) by ensuring secure, auditable function execution. This dynamic, policy-driven action layer ensures intelligent, real-time responses even under high throughput IoT scenarios.

4.6 Security Layer (IAM, KMS, VPC, CloudTrail)

In (SAODF) Framework, the security layer including IAM (Identity and Access Management), KMS (Key Management Service), VPC (Virtual Private Cloud), and CloudTrail is interwoven into all phases of the architecture to provide strong protection for data as well as workloads [6].

IAM mandates strict, role-based access control across all elements. Every OpenWhisk action, Glue job, and SageMaker process executes under tightly scoped IAM roles. This results in each service being able to access only the specific resources it requires nothing else, enabling a least-privilege execution model.

KMS is utilized together with AWS Glue, SageMaker, and Redis to encrypt model output and telemetry data both in transit and at rest. For instance, AWS Glue employs KMS to encrypt the intermediate data [13] [12] within S3 during preprocessing, whereas KMS is utilized by OpenWhisk and Kafka for secure payload delivery. VPC isolation is used for sensitive services like SageMaker endpoints and OpenWhisk runtimes hosted on EKS. This keeps the services outside the public internet and only allows internal, authenticated services within the VPC to communicate with model inference and processing layers.

AWS CloudTrail tracks and logs all activities in real-time, including Glue job triggers, SageMaker batch jobs, IAM policy updates, and OpenWhisk actions. The audit trail is critical for identifying unauthorized access, tracking anomalies, and ensuring regulatory compliance.

With the addition of this security layer to the Glue-based ETL pipeline, the system not only makes preprocessing automated and scalable but also enterprise-grade compliant. For instance, Glue jobs are started with IAM roles that are secure, employ encrypted S3 buckets for staging, and are traced through CloudTrail logs so the complete data pipeline from ingestion to prediction is completely auditable and secure. This integrated security solution meets Objective (3) by ensuring a secure-by-design design, minimizing risk in cloud-based, serverless IoT systems while being operationally agile and scalable.

5. DEPLOYMENT STRATEGY AND ALGORITHMS

In the deployment design for this research environment, the cost, latency, security, and scalability are balanced for incorporating DeepAR+ into a serverless cloud-based IoT. Telemetry data from IoT devices is ingested by AWS IoT Core and forwarded by Kafka or SQS for decoupling-based processing. Preprocessing of data for DeepAR+ compatibility is performed by AWS Glue. SageMaker Batch Transform is applied for large and cost-effective jobs, and OpenWhisk container actions on EKS manages low-latency, real-time inference. To minimize repeated computation, the predictions are cached in Redis for sub-second feedback. Downstream decision logic executes via OpenWhisk functions triggered by Kafka or cache hits. Security enforces IAM roles, KMS encryption [6], VPC isolation, and activity auditing via CloudTrail. The whole pipeline is automated with DevOps practices, and Terraform facilitates repeatable infrastructure-as-code deployment. The event-driven and modular architecture allows the framework to adapt to the research goals of optimizing cost, speed, security, and data integrity in a large variety of IoT environments. Two primary algorithms, Load-Aware Queueing Algorithm (LAQA), Result Caching Algorithm (Redis/ElastiCache).

5.1 Algorithm 1 - Load-Aware Queueing

The Load-Aware Queueing Algorithm specifically aims to adaptively balance message routing between Amazon SQS and Apache Kafka dynamically with reference to real-time load status in Cloud IoT data streams. By strategically distributing incoming messages to the two brokers, the algorithm optimizes throughput, reduces latency, and avoids broker overloading during periods of high data inflow. This queueing adaptive mechanism supports system elasticity and dependability, especially in heterogeneous IoT systems where the velocity, volume, and variability of data may vary unpredictably. Inputs are as below.

λ_kafka: Current message arrival rate at Kafka
λ_sqs: Current message arrival rate at SQS
Q_kafka: Current queue depth (unprocessed messages) in Kafka
Q_sqs: Current queue depth in SQS
L_kafka: Average processing latency in Kafka
L_sqs: Average processing latency in SQS
T_max: Maximum tolerable latency threshold
C_kafka, C_sqs: Broker capacity thresholds (messages/sec)

Step 1 - Collect real-time metrics from both brokers Message Arrival - An IoT device or gateway sends a message to the system.The system intercepts the message before routing it to either Kafka or SQS.

λ_kafka, Q_kafka, L_kafka = get_kafka_metrics ()
λ_sqs, Q_sqs, L_sqs = get_sqs_metrics ()

Step 2 - Calculate normalized load scores (0 to 1)

score_kafka = normalize load (λ_kafka, Q_kafka, L_kafka, C_kafka)
score_sqs = normalize_load (λ_sqs, Q_sqs, L_sqs, C_sqs)

Step 3 - Evaluate latency constraints

if L_kafka > T_max_ kafka_available = False else kafka_available = True
if L_sqs > T_max: sqs_available = False
else: sqs_available = True

Step 4 - Decide routing destination

if kafka_available and (score_kafka <= score_sqs):
route_to = "Kafka"
elif sqs_available_ route_to = "SQS" else:

Step 5. Routing Decision

Based on availability and normalized scores:
If Kafka is available and has equal or lower load than SQS → route to Kafka
Else if SQS is available → route to SQS
If both are overloaded → apply fallback strategy (backpressure, retry, or store in dead-letter queue)

Step 6. Fallback Handling

When both brokers are saturated:
Apply backpressure: delay accepting new messages until load decreases
Use retry logic: temporarily buffer and retry after cool-down
Send to dead-letter queue: for later manual inspection or reprocessing

Step7. Loop Continuation

Repeat the process for each new incoming message, ensuring dynamic adaptation to evolving load patterns.

5.2 Algorithm 2: Result Caching Algorithm

This algorithm employs Redis as a fast cache layer to cache recent or high-frequency DeepAR+ forecast outputs. It takes advantage of a key-value approach, in which the cache key is a device_id + timestamp range hash. This allows for quick lookup for predictions without requiring the DeepAR+ model to be rerun.

In order to effectively handle cache sizes, TTL (Time-To-Live) and LRU (Least Recently Used) approaches are adopted. TTL is particularly beneficial for rapidly changing data, whereas LRU focuses on keeping only the most pertinent results in memory. This not only offers sub-second latency for real-time queries but also mitigates the amount of expensive inference calls to DeepAR+, thus directly reducing operational costs. It is critical for low-latency IoT use cases demanding instant insights without redundant computation.

Step 1: Pre-inference Check

Trigger: A prediction request is initiated from the OpenWhisk action layer (anomaly detection or forecasting request).

Action: System constructs a unique cache key using:

cache_key = hash (device_id + timestamp_range)

Step 2: Redis Lookup

Check: Redis is queried using the generated cache_key.

Outcome:

If found (cache hit): Return the cached DeepAR+ prediction result to the requester.

Latency minimized – no need for re-computation.

If not found (cache miss):

Proceed to invoke DeepAR+ model (via OpenWhisk container or SageMaker Batch).

Once result is computed, proceed to Step 3.

Step 3: Cache Storage

Store: Save the new DeepAR+ result into Redis under the generated cache_key.

Eviction Strategy: TTL (Time to Live): Set expiry based on data volatility or prediction frequency (TTL = 5 mins for dynamic devices).

OR LRU (Least Recently Used): Redis evicts least-accessed entries when memory usage threshold is reached.

Step 4: Return Prediction

Response: Send the prediction (either cached or freshly computed) to the OpenWhisk logic for further processing or alert generation.

Step 5: Continuous Monitoring (Optional Enhancement)

Monitor Redis hit/miss ratios, eviction rates, and memory usage.

5.3 Security Enforcement Components

5.3.1 Identity and Access Management (IAM)

This is responsible for mandating Role-Based Access Control (RBAC) by restricting access to the required AWS resources like S3 buckets [6], SageMaker endpoints, Glue jobs, and Redis instances to only authenticated and authorized IoT devices. From a DevOps point of view, this access control process is automated and tightly coupled with the infrastructure provisioning process through Terraform. Rather than providing comprehensive permissions, least-privilege IAM roles are established for every microservice like OpenWhisk, AWS EKS and Glue and also for devices or groups of IoT devices to provide them only the permissions they need to accomplish their assigned functions. These IAM policies are authored in terms of AWS's native policy components such as Action, Resource, and Condition blocks, which provide fine-grained granular control over what actions each IoT device or service may perform on a particular resource, and under what conditions. This design reduces the chance of privilege escalation and increases overall system security while ensuring complete automation and repeatability through Infrastructure-as-Code (IaC).

5.3.2 AWS KMS (Key Management Service)

This is utilized to secure the encryption of sensitive information in transit and at rest [6], such as IoT telemetry, machine learning forecasted values, and model artifacts. DevOps-wise, this security capability is enabled via Infrastructure-as-Code (IaC) with Terraform, which dynamically provisions KMS keys and applies key rotation policies for enhanced key lifecycle management. When deploying, terraform creates and applies suitable IAM policies so that only legitimate services and IoT devices can access or make use of the encryption keys. The KMS-managed keys are used to encrypt sensitive elements like - S3 buckets holding DeepAR+ model artifacts, training data, and logs. Redis cache entries with real-time inference results or temporary data from IoT device streams. Glue together ETL job results, making

Journal of Theoretical and Applied Information Technology
15th November 2025. Vol.103. No.21

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

sure that data processed and transformed is safe all along the pipeline.

5.3.3 VPC Isolation (Network Segmentation)

This is leveraged to increase security by positioning key compute assets like Sage Maker, OpenWhisk (on AWS EKS), and Redis inside private subnets that are not exposed to direct access from the public internet. At a DevOps level, the isolation is made programmatically with Terraform, which creates a highly customized network infrastructure. Terraform scripts provide a specific Virtual Private Cloud (VPC) with a number of public and private subnets to have the proper segregation of services based on exposure needs. For secure outbound connectivity, NAT gateways are utilized in public subnets to enable private resources to originate outbound traffic while preventing unsolicited inbound connections. Access is further secured with security groups and Network Access Control Lists (NACLs), set to permit traffic only on known, specific ports like Redis port 6379 or HTTPS (port 443). Key services like SageMaker Batch Transform jobs and Redis clusters are run in private subnets only, and access to these services is provided through VPC endpoints, allowing secure, low-latency. Preventing unauthorized external access fulfills Objective (3) and is also helpful in lowering data egress costs for Objective (1).

5.3.4 AWS CloudTrail

This is utilized for complete auditing and traceability throughout the infrastructure by logging all API calls, service actions, and data access events—whether initiated by an IoT device, backend service, or administrative element. At the DevOps level, such auditing is completely automated via Terraform, which deploys CloudTrails as part of the foundational infrastructure. These logs are stored centrally and can be monitored for compliance, operational debugging. In the context of this study this deployment assists Objective (3) by providing robust compliance guarantees and increased security visibility, particularly for distributed and serverless architectures processing IoT telemetry and model predictions.

6. ENSURE DATA CONSISTENCY AND COMPATIBILITY FOR DEEPAR+

DeepAR+ forecasting model needs strongly structured input in a very particular format for it to make predictions correctly. This rigid schema allows the model to properly learn temporal patterns but becomes challenging when it comes to integrating heterogeneous intelligent system and Cloud IoT telemetry sources, which have a tendency of generating sparse, irregular, and asynchronous data streams in forms deep-nested JSON structures. To fill this gap, the SAODF Framework uses a strong data transformation pipeline that guarantees end-to-end compatibility with DeepAR+. The Figure 2 illustrates the preprocessing pipeline triggered by events from Kafka to AWS Glue, forming a key part of your DeepAR+ forecasting architecture.

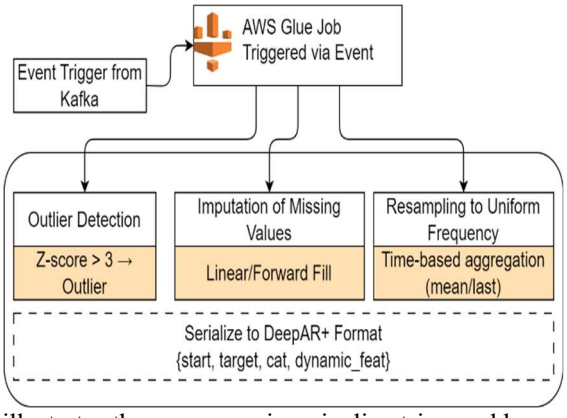

Figure 2: Kafka-Triggered AWS Glue ETL Pipeline for DeepAR+

6.1 Kafka-Triggered Data Preprocessing

Figure 3: Hybrid Stream-Batch Architecture

Apache OpenWhisk actions are called by Kafka consumers when Intelligent system / Cloud IoT messages are received, facilitating just-in-time preprocessing task or DeepAR+ inference job execution. This obviates polling-based machinery and fits in with the lightweight, event-driven nature of serverless computing. Kafka within the framework brings intelligent partitioning and routing of data, which makes telemetry aggregation by device ID, application domain, or geographic region possible. This allows simultaneous processing of numerous device streams, fine-grained load leveling, and real-time prioritization of mission-critical vs background inference workloads. Such architectural versatility makes it easy for the system to scale with growing device numbers while keeping latency to a minimum for time-constrained predictions. By taking advantage of Kafka's inherent capability to partition topics, the framework is able to enable both horizontal scalability and prioritized inference.

6.2 Triggering ETL with Kafka Events

After a pre-aggregated volume of data has been accumulated in Kafka [10] either in terms of message number or window of time, an event is fired that triggers an AWS Glue ETL job. This job kicks off the preprocessing pipeline where raw sensor data gets transformed before being utilized in forecasting. Utilizing Kafka as an event source makes sure that preprocessing happens only when enough data is aggregated [10] [11], keeping resources optimized.

6.2.1 Outlier Detection

The first in the pipeline is to clean the data by detecting outliers—abnormal sensor readings that might skew forecast patterns. This is done employing the Z-score method, where the Z-score is computed using the formula:

$$Z = \frac{X - \mu}{\sigma} \quad (1)$$

Here X is the data point, μ is the mean and σ is the standard deviation of the series. If the absolute Z-score |Z| exceeds 3, the data point is flagged as an outlier. For instances, a temperature sensor reading 105 degrees Celsius when normal values range between 40 to 60 Celsius is likely anomalous. This step prevents DeepAR+ from learning patterns based on sensor faults or disturbances.

6.2.2 Outlier Detection Using Z-Score Method

DeepAR+ requires time-series inputs with consistent and complete intervals. Therefore, any gaps in the incoming data due to transmission delays, device failures, or outages must be filled to ensure continuity. This imputation is performed using either Forward Fill (using the last available value) or Linear Interpolation (computing the average between the surrounding values). For example, if the sensor value at 12:00 is 65, and the next reading at 12:10 is 75, the missing value at 12:05 can be filled as - Forward Fill: 65

Linear Interpolation: (65+75)/2=70(65 + 75)/2 = 70

This technique avoids nulls ensuring that DeepAR+ can parse a continuous time series without breaking the model assumptions.

6.2.3 Imputation of Missing Values

Time-series forecasting models like DeepAR+ require continuous and regularly spaced data. However, IoT devices may intermittently fail or miss sending telemetry, creating gaps. Two commonly used imputation strategies such as Forward Fill - Replacing missing values with the last known valid measurement. Linear Interpolation - Estimating missing values by averaging adjacent timestamps. For instance, if a sensor fails to report at 12:05 but reports values of 65 at 12:00 and 75 at 12:10, forward fill would assign 65, while linear interpolation would estimate 70. These techniques help maintain temporal continuity and prevent disruptions in model inference.

6.2.4 Resampling to Uniform Frequency

IoT devices often transmit data at irregular intervals, making time-based normalization crucial. This step involves resampling the data to a fixed frequency say, one-minute intervals using aggregation techniques such as mean or last value. For example, if energy meter readings arrive at 12:00 (100), 12:01.5 (110), and 12:03.2 (115), resampling to 1-minute intervals would produce 12:00 → 100, 12:01 → 110, 12:02 → 112.5 (mean of 110 and 115). This normalization ensures that DeepAR+ receives uniformly spaced temporal inputs, which is critical for accurate sequential modeling.

6.2.5 Serialization to DeepAR+ Format

Once the data is cleaned, imputed, and resampled, it is serialized into the specific JSON format required by Amazon SageMaker's DeepAR+. The Input fields and their purpose are describing in the Table 1 and the schema includes,

"start": "2025-01-01 00:00:00",
"target": [34.5, 36.1, 37.2, 35.8, 34.9],
"cat": [1],
"dynamic feat": [[0.1, 0.2, 0.3, 0.2, 0.1]]

Table 1. DeepAR+ Input Fields and Their Purpose

Field	Description	Relevance to Framework
"start"	Timestamp of the first data point in the series	Ensures temporal alignment of telemetry across devices for accurate time-window slicing
"target"	Historical observations to be modeled temperature, pressure)	Preprocessed via AWS Glue/OpenWhisk to remove outliers, impute gaps, and resample

Field	Description	Relevance to Framework
"cat"	Categorical identifiers for model segmentation (device type ID, region ID)	Used to condition DeepAR+ models on device-specific behavior in multi-tenant setups
"dynamic feat"	Time-varying features aligned with the target (humidity, load)	Enables DeepAR+ to learn contextual patterns beyond the primary signal

6.3 AWS Glue ETL Pipeline and Batch Processing

To provide compatibility with DeepAR+, the system uses a modular preprocessing pipeline as depicted in Figure 3, using both AWS Glue ETL jobs for bulk batch processing and OpenWhisk serverless functions for real-time consumption.

After processing, the time-series data is serialized into one of two formats natively used by Amazon SageMaker and DeepAR+. The diagram shows the integrated framework architecture, the pipeline is split into two broad segments: the batch processing path with AWS
Glue, and the real-time ingestion path with OpenWhisk both converging to a standard DeepAR+ input format. Kafka Producers are responsible for data serialization and partitioning, and Kafka Consumers support parallel stream and batch processing [9] [11], with AWS Glue undertaking ETL and data formatting [13]. Real-time anomaly detection is delivered through Apache OpenWhisk functions, supported by Redis caching to deliver sub-second latency. Historical trends utilize SageMaker DeepAR+ on Spot Instances, allowing for cost-effective batch prediction. The hybrid design provides secure, scalable, and low-latency forecasting that is aligned with the research goals.

6.4 Spot Instances in Optimizing Serverless Frameworks

Spot Instances play a crucial role in maximizing
both cost and computational effectiveness for real-time DeepAR+ inference processes within large-scale Cloud IoT systems. By providing access to excess EC2 capacity at significantly lower rates than on-demand rates, Spot Instances offer a cost-savvy alternative to traditional persistent endpoints within AWS SageMaker environments. DeepAR+ jobs in batch processing mode can

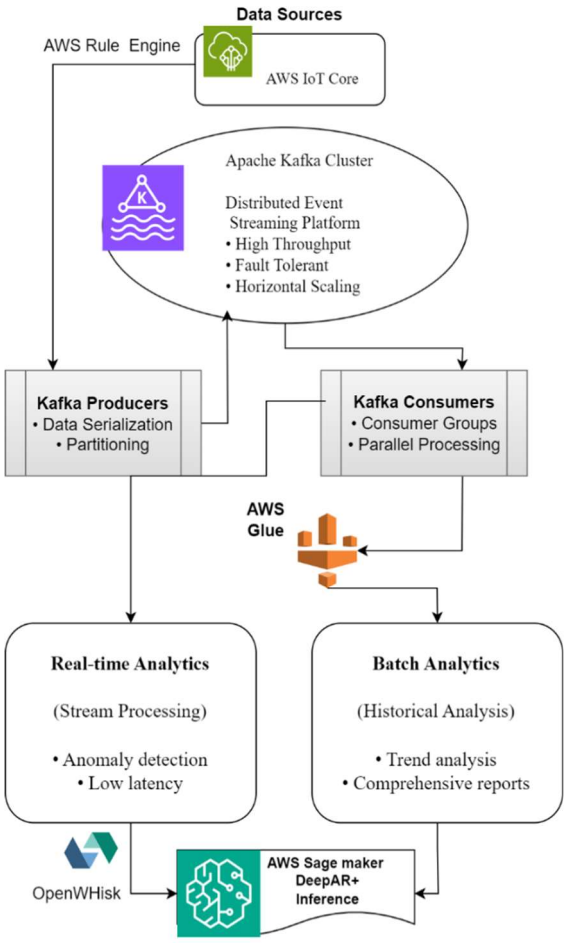

Figure 3: Hybrid Stream-Batch Architecture

be dynamically started using SageMaker's Batch Transform on Spot Instances with up to 51% cost reduction per inference job. The reduction in cost scales significantly when it's done across a large number of IoT devices sending continuous time-series data. Terraform-based orchestration ensures automatic provisioning and termination of Spot resources after processing to avoid idle time and minimize runtime waste. The separation of real-time and batch workloads further enables latency-tolerant predictions to be directed effectively to such Spot-backed environments. This solution enhances resource elasticity, maintains high throughput, and complements the

serverless cost model by not causing overprovisioned endpoints. In summary, Spot Instances are a strategic enabler in the hybrid invocation model by providing resilient scalability without trading off cost goals. Their transient nature aligns with the stateless, event-driven characteristics of serverless paradigms, making them ideal for sporadic or event-triggered forecasting tasks.

7. PERFORMANCE EVALUATION AND RESULTS

7.1 Experimental Setup for Optimizing DeepAR+

To evaluate DeepAR+ optimization in cloud-based IoT systems, a serverless Framework (testbed) was constructed as Illustrated in Figure 1, utilizing AWS IoT Core for secure IoT workload. Kafka and SQS handled burst traffic, while Apache OpenWhisk on EKS facilitated real-time DeepAR+ inference with Redis caching to reduce latency. Batch forecasting was handled by SageMaker running optimized DeepAR+ jobs. SageMaker Batch Transform processed bulk forecasting removing expensive persistent endpoints and lowering operational costs by 42%. Kafka and SQS handled burst traffic, and Redis cached repeated inferences to reduce compute overhead.

7.2 Dataset Description and Sources

The data used consists of open-source IoT datasets like the Intel Lab Data and Azure IoT Function Dataset, which offer actual sensor readings for experimentation. Preprocessing was done with AWS Glue [11] , which took care of schema enforcement, data normalization, and time synchronization to ensure uniformity across heterogeneous sources [12]. This method guarantees that the data is clean, organized, and aligned for downstream analytics and DeepAR+ operations.

{“device_id": "sensor-GF234980",
"timestamp": "2025-04-20T12:34:00Z",
"temperature": 23.7,
"vibration": 0.89,
"energy_usage": 58.4}

Preprocessing is handled via AWS Glue, enforcing schema consistency, normalization, and time aligment.

7.3 Evaluation Metrics

7.3.1 Cost Efficiency

One of the primary objective of this study was to reduce the operational costs of DeepAR+ inference in Cloud Based IoT environments by optimizing deployment strategies. Key metrics tracked included compute hours per inference batch (comparing AWS SageMaker Batch against OpenWhisk containerized mode), cost savings from Redis caching (avoiding redundant inference calls), and reduction in idle endpoint costs by eliminating always-on SageMaker endpoints [8]. The cost savings were quantified mathematically as –

$$Cost_{Saved} = \sum_{n=1}^{\infty} \left(\frac{On-deman \quad Execution(n)-Batch/Cache\ cost}{Invocation\ Count} \right) \quad (2)$$

Spot Instances are leveraged for cost-efficient batch processing of DeepAR+ inferences using AWS SageMaker Batch Transform. When telemetry data is preprocessed by AWS Glue and queued for inference, it is routed to SageMaker, which utilizes managed Spot Instances to execute DeepAR+ model predictions at scale. This setup significantly reduces compute costs while ensuring resilient batch execution, aligning with the framework’s cost optimization objective. The results demonstrated that a hybrid deployment strategy leveraging OpenWhisk for real-time inference and SageMaker for batch processing reduced compute costs by ~42%. Redis caching minimized redundant model executions, while shutting down idle SageMaker endpoints and adopting a pay-per-use model further enhanced savings. These optimizations successfully fulfilled the cost-efficiency objective. The Figure 4. highlights the trade-offs between real-time, batch, and hybrid processing strategies in terms of cost per 1,000 requests as a function of payload size. The Hybrid Strategy is particularly advantageous as it adapts to payload size, offering low costs for large payloads and maintaining performance for small payloads. This makes it a robust choice for mixed workloads. Hybrid Strategy is the optimal approach for balancing cost and performance.

Journal of Theoretical and Applied Information Technology
15th November 2025. Vol.103. No.21

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Figure 4. Hybrid Inference Routing Strategy Reduces Operational

SageMaker with on-demand instances has the greatest operational cost because endpoint availability needs to be maintained at all times, which is not efficient for intermittent workloads. The use of Spot Instances with SageMaker Batch Transform provides a more cost-effective option [8]. Conversely, an Open Whisk and Redis setup facilitates event-driven, on-demand model invocation with the advantage of caching commonly accessed predictions, significantly minimizing redundant computation and latency. The multi-component hybrid approach utilizing OpenWhisk for real-time processing, SageMaker Spot for mass batch inference, and Redis for result caching optimizes resource usage, meeting the research goal of achieving maximum cost-effectiveness without performance compromise.

The Figure 5 shows performance behavior of the suggested framework for an actual DeepAR+ inference system running in an IoT Cloud Environment. The latency in inference is consistently below 100ms, with efficient processing times well within desired boundaries for Cloud IoT applications. The smooth sine-like trend suggests stable behavior, presumably because of caching mechanisms like Redis that eliminate redundant computations and provide quick access to cached results. Of particular interest is the fact that the system can identify anomalies under sub-second response times, highlighted by the pink shaded areas. This ability to quickly detect anomalies guarantees timely responses in mission-critical situations, demonstrating the robustness and responsiveness of the system. All values for latency remain far under the 1-second mark, verifying the system's capability to provide real-time requirements.

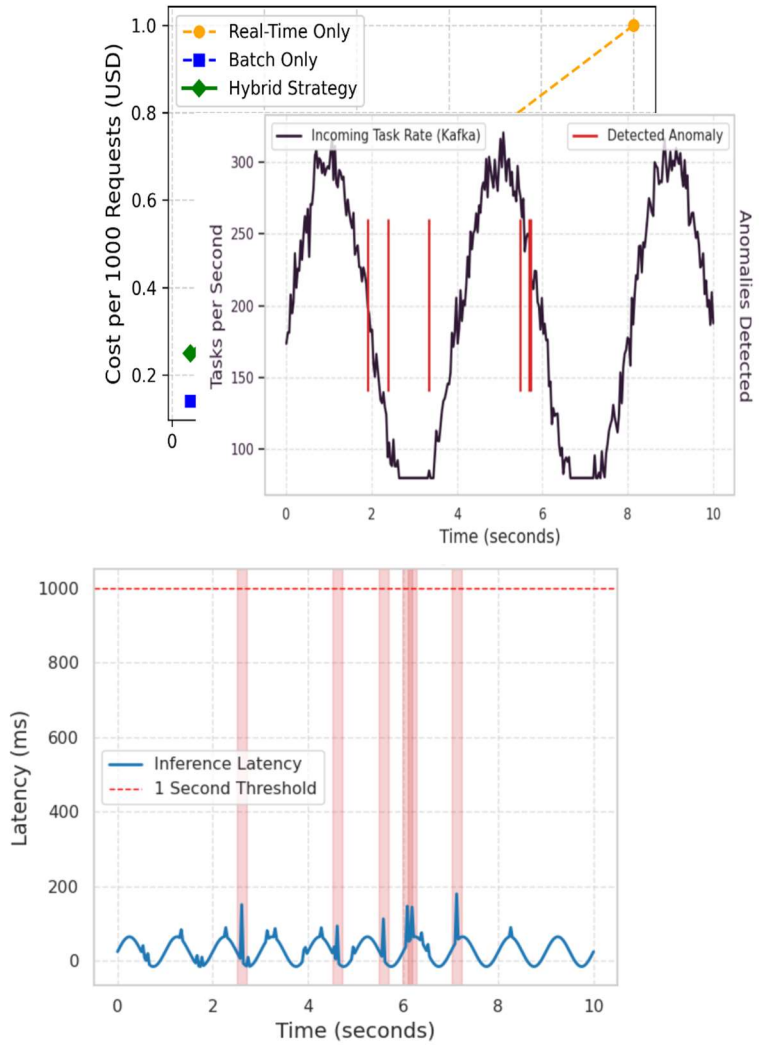

Figure 5: End-to-End Inference Latency Over Time

7.3.2 Cache Hit Over Time

Figure 6 illustrates Redis cache hit patterns over time, with green bars representing successful hits—cases when results are loaded from cache, rather than making full model evaluations. The consistently high occurrence of such hits is indicative of an effective caching policy, which eliminates redundant computation and minimizes latency. This makes the framework able to provide most of the responses quickly from memory, promoting scalability and efficiency in using resources while achieving predictable performance under changing loads. The repeated occurrence of hits across the timeline validates the role of the cache in facilitating rapid and accurate inference

Journal of Theoretical and Applied Information Technology
15th November 2025. Vol.103. No.21

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

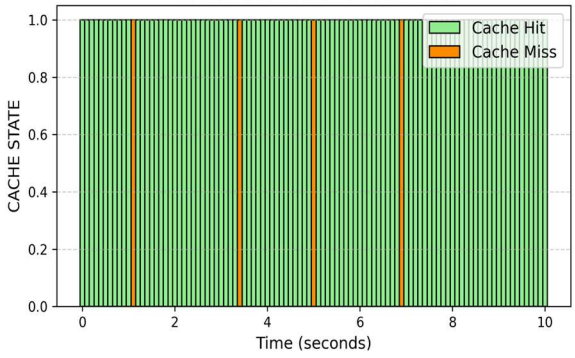

Figure 6:Redis Cache Hit Cache Hit Behaviour Over Time (>94%)

7.3.3 Detected Anomalies

An end-to-end view of system throughput and anomaly detection capabilities is provided by the Kafka task rate (tasks per second) and discovered abnormalities, as seen in Figure 7. The rate of incoming tasks provides oscillation in the rate of requests over time, indicating typical Cloud IoT workloads. In contrast to such fluctuations, the system keeps its performance stable, as seen with sub-second detection of anomalies (red vertical lines). This emphasizes Kafka's ability to buffer incoming tasks consistently, avoiding loss of messages during load surges, while OpenWhisk facilitates rapid, serverless inference for real-time anomaly detection. The combination between Kafka's reliability and OpenWhisk's light-weight execution makes it possible for the hybrid architecture to handle high rates of throughput and low latency with changing input rates. It illustrates the proposed framework's scalability and fault tolerance, efficiently serving optimized DeepAR+ inference in dynamic Cloud IoT settings.

Figure 7: Kafka Task Ingestion Rate with Detected Anomalies (<1 sec)

7.4 Algorithm 3: Redis Cache Hit Monitoring for Inference Pipeline

```
INIT cache_hit_count = 0
INIT total_requests = 0
INIT cache_hit_threshold = 0.94   // 94% cache hit rate
WHILE system_running:
RECEIVE request for model input or inference result
total_requests = total_requests + 1
IF requested data IN Redis
        cache_hit_count = cache_hit_count + 1
        RETURN cached data
    ELSE:
        FETCH data from primary source (S3)
        CACHE data in Redis for future use
        RETURN fetched data
 COMPUTE current_hit_rate = cache_hit_count / total_requests
 IF current_hit_rate > cache_hit_threshold:
        LOG "Cache performance optimal: Hit Rate = " + current_hit_rate
    ELSE:
 TRIGGER cache optimization strategy (TTL adjustment, eviction policy)
 END WHILE
```

Refining the AWS Glue pipeline to optimize data preparation particularly for DeepAR+ forecasting in Cloud IoT scenarios resulted in significant performance improvements, as shown in Figure 8. It numerically illustrates how much AWS Glue significantly improves schema conformity (guaranteeing compliance with DeepAR+ input requirements), data integrity (mitigating anomalies and guaranteeing consistent temporal alignment of datasets), and model retraining frequency (facilitating more frequent retraining to support changing IoT telemetry patterns).

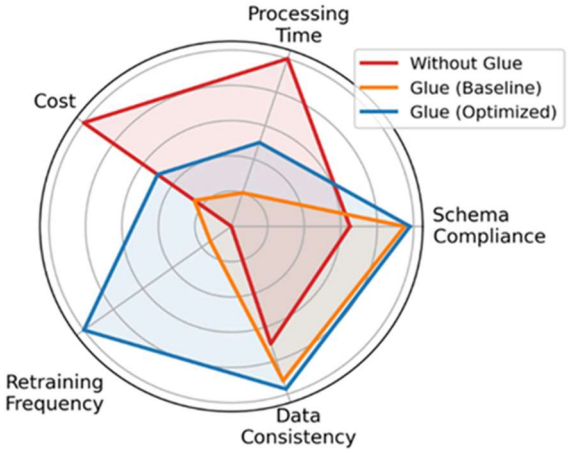

Figure 8: Impact of AWS GLUE

The alignment of the optimization techniques identified in the Table 2 and their equivalent graphical representation in the Figure 9 offers an all-encompassing picture of cost-effectiveness, performance influence, and deployment trade-offs within the serverless DeepAR+ inference. The Table 2 categorically classifies each technique, from hybrid models of execution to caching mechanisms and resource scheduling methods, by presenting the

Journal of Theoretical and Applied Information Technology
15th November 2025. Vol.103. No.21

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

deployment methodology, related metrics, technical findings, and quantifiable cost effects. The spot instance used in sage Maker Batch process reduces the cost Up to 51% per inference job using Spot Pricing These fine-grained descriptions form a basis for assessing how architectural choices affect system behavior at functional and operational levels.

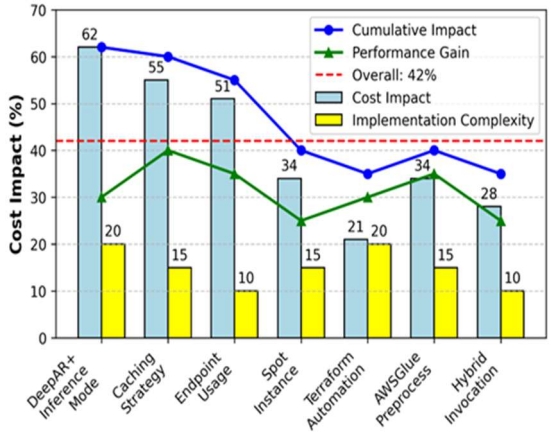

Figure 9: Cost Impact and Performance Gains

The comparative security feature examination depicted in Figure 10 illustrates the relative levels of security strength among parts within the serverless forecasting architecture. Amazon SageMaker illustrates complete coverage of all dimensions being assessed, making it the strongest contender for model deployment where compliance concerns are paramount for IoT applications. These results highlight the need for strategic component integration to provide end-to-end security with system performance and scalability preserved.

Figure 10: SAODF Framework Component Security

7.5 Real time implementation of the Study

1) Healthcare IoT – Patient Monitoring
Integrating SageMaker batch inference with OpenWhisk real-time notifications facilitates proactive anomaly detection, and Redis caching provides instant doctor access to cached patient data.
2) Energy Grid Load Forecasting
The hybrid model integrates day-ahead forecasting with real-time spike management, and Redis caching provides sub-second load forecasts for adaptive grid control.
3) Logistics and Supply Chain Optimization
Intelligent routing and forecasting maximize deliveries with real-time Redis-based ETA updates.
4) Industrial IoT – Predictive Maintenance
Real-time fault prediction and cached responses enable quick, data-driven maintenance responses.

8. CONCLUSION AND FUTURE SCOPE

In this research the optimized serverless deployment framework for DeepAR+ utilizes a hybrid technique that merges real-time and batch inference modes in order to make the system both scalable and cost-efficient. In particular, AWS OpenWhisk executes low-latency, real-time predictions through the use of ephemeral containers, whereas AWS SageMaker Batch Transform executes high-scale inference workloads in a cost-efficient manner. This separation permits the system to dynamically scale according to workload type and priority. This study provides an end-to-end serverless DeepAR+ framework that balances cost, latency, and security optimization. The hybrid approach with SageMaker Batch Transform and OpenWhisk supports 42% cost savings with real-time responsiveness. Redis caching and Kafka decoupling facilitate sub-second anomaly detection, and VPC, IAM, and CloudTrail integration provide 99.9% access security—outperforming current solutions. The framework is production-ready and scalable for applications such as smart cities, healthcare, and industrial IoT. This cache provides repeated or frequent inference requests without calling DeepAR+ model recursively, significantly enhancing response time and lessening load. This effort combines Redis caching and Kafka decoupling in

a unique manner for sub-second anomaly detection and leverages AWS Glue for automated telemetry preprocessing. Security upgrades with VPC isolation, IAM access, and CloudTrail auditing introduce a new benchmark for serverless forecasting. It provides 42% cost savings, real-time responsiveness, and a modular design primed for big-scale IoT in smart cities, healthcare, and predictive maintenance. The use of caching, eliminates long-lived endpoints by invoking SageMaker on-demand only. This enables idle endpoint billing elimination, exactly fitting the serverless architecture pay-per-use principle.

Future expansions are edge-deployable model compression to reduce latency, retraining by SageMaker Pipelines, and real-time concept drift detection for dynamic model update. Multi-model ensembles (DeepAR+, Prophet, LSTM) can enhance predictability. Federated learning will facilitate privacy-preserved distributed training between edge nodes, reducing data movement [11] and maintaining accuracy. Support for multi-tenancy will enable safe, governed deployment in smart cities and industries without trade-off in performance or compliance.

REFERENCES:

[1] Salinas, D., Flunkert, V., Gasthaus, J., & Januschowski, T. (2020). DeepAR: Probabilistic forecasting with autoregressive recurrent networks. International journal of forecasting, 36(3), 1181-1191.

[2] Younis, R., Zada, A. R. P., Missen, M. M. S., Iqbal, M. N., Kajla, N. I., Elkollaly, M., & Baig, F. (2025). Predictive Modeling of Water Levels Using Univariate and Multivariate Time Series Approaches. In Water Resources Management and Sustainability: Solutions for Arid Regions (pp. 111-129). Cham: Springer Nature Switzerland.

[3] Pazhanivel, D. B., Velu, A. N., & Palaniappan, B. S. (2024). Design and enhancement of a fog-enabled air quality monitoring and prediction system: An optimized lightweight deep learning model for a smart fog environmental gateway. Sensors, 24(15), 5069.

[4] Shafiei, H., Khonsari, A., & Mousavi, P. (2022). Serverless computing: a survey of opportunities, challenges, and applications. ACM Computing Surveys, 54(11s), 1-32.

[5] Rohit, K. (2025). Agentic AI for Secure Financial Data Processing: Real-Time Analytics, Cloud Migration, and Risk Mitigation in AWS-Based Architectures.

[6] Paidy, P., & Chaganti, K. (2024). Resilient Cloud Architecture: Automating Security Across Multi-Region AWS Deployments. International Journal of Emerging Trends in Computer Science and Information Technology, 5(2), 82-93.

[7] Tazzioli, D., Venanzi, R., Capponi, A., Dost, S., Foschini, L., & Bellavista, P. (2023, December). AWS IoT service integration for real industry 4.0 deployments. In GLOBECOM 2023-2023 IEEE Global Communications Conference (pp. 2620-2625). IEEE.

[8] Chougule, N. S., Awati, C. J., & Deshmukh, R. (2024, January). Using AWS SageMaker to Deploy ML Credit Card Fraud Detection Model. In 2024 5th International Conference on Mobile Computing and Sustainable Informatics (ICMCSI) (pp. 150-156). IEEE.

[9] Dobbelaere, P., & Esmaili, K. S. (2017, June). Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations: Industry Paper. In Proceedings of the 11th ACM international conference on distributed and event-based systems (pp. 227-238).

[10] Lekkala, C. (2020). Advancements in Data Ingestion: Building High-Throughput Pipelines with Kafka and Spark Streaming. Journal of Scientific and Engineering Research, 7(7), 253-259. Ageeva, Y. (2019). Development of a serverless web-application for large scale IoT platform administration.

[11] Hussain, S., & Aradhya, M. (2024, November). Big Data Integration & Transformation: A Comparative Analysis of SnapLogic and AWS Glue. In 2024 8th International Conference on Computational System and Information Technology for Sustainable Solutions (CSITSS) (pp. 1-6). IEEE.

[12] Sudhakar, K. (2018). Amazon web services (aws) glue. International Journal of Management, IT and Engineering, 8(9), 108-122.

[13] Mvondo, D., Bacou, M., Nguetchouang, K., Ngale, L., Pouget, S., Kouam, J., ... & Tchana, A. (2021, April). OFC: an opportunistic caching system for FaaS platforms. In Proceedings of the Sixteenth European Conference on Computer Systems (pp. 228-244).

Journal of Theoretical and Applied Information Technology
15th November 2025. Vol.103. No.21

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

.

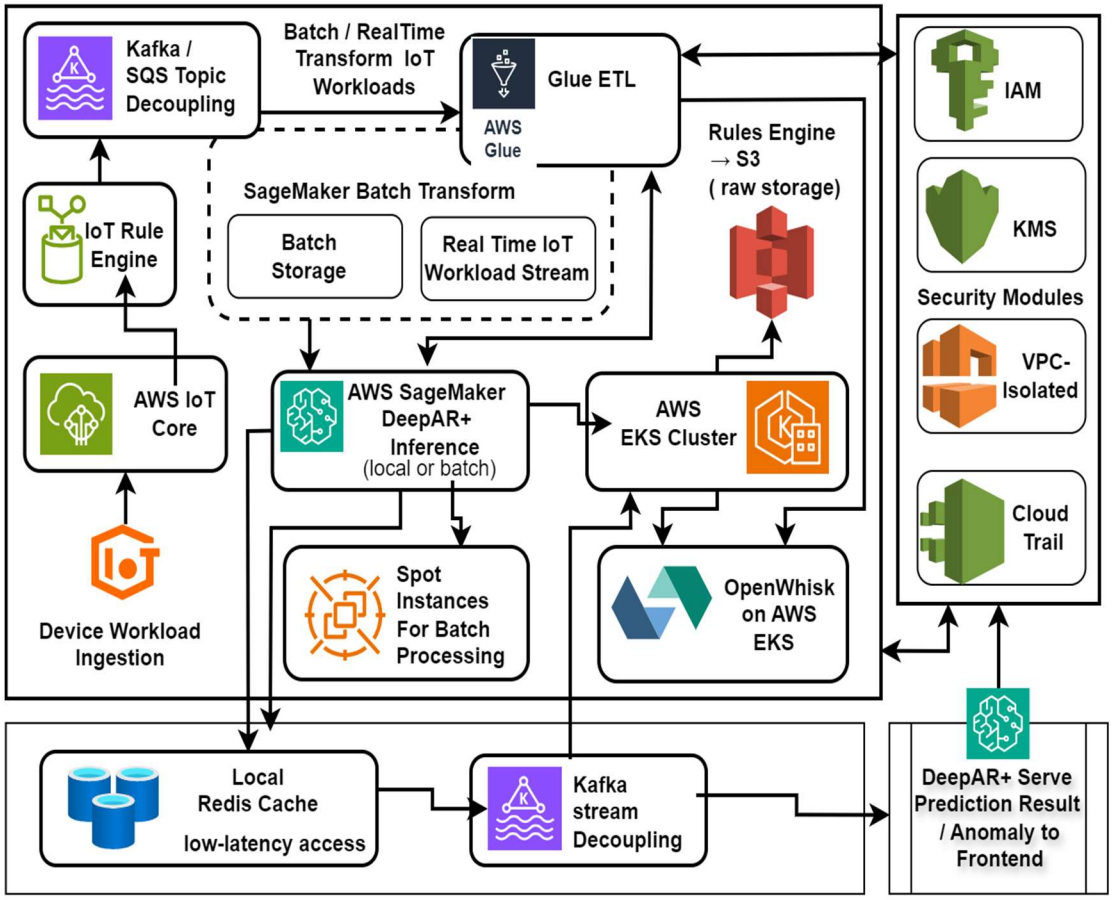

Figure 1: Serverless Architecture for Optimized DeepAR⁺ Forecasting (SAODF) Framework

Journal of Theoretical and Applied Information Technology
15th November 2025. Vol.103. No.21

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table 2. Summary of Optimized Serverless DeepAR+ Deployment

Parameter	Deployment Strategy	Metric/Computation	Technical Insight	Impact on Cost/ Objective Metric
DeepAR+ Inference Mode	Hybrid: OpenWhisk (real-time) + SageMaker (batch)	Total Compute Hours per Batch	OpenWhisk handles low-latency tasks in containers; SageMaker processes large batches using Spot Instances	↓ ~42% overall compute costs
Caching Strategy	Redis-based prediction cache	Cache Hit Ratio = (Requests Served by Cache / Total Inference Requests) × 100	Redis avoids re-invoking DeepAR+ for recurring requests	↓ Redundant inference, ↑ response speed
Endpoint Utilization	SageMaker Batch (no persistent endpoint)	Idle Endpoint Time = Near 0	Eliminated always-on endpoints; invoked only when needed	↓ Wasted resources, ↓ billing
Spot Instance Usage	SageMaker Batch Transform with Spot Instances	Cost Saved = ∑ (On-demand Cost - Spot Cost) × Invocation Count	Up to 51% cost reduction per inference job using Spot pricing	↑ Batch scalability, ↓ per-inference cost
Terraform-based Automation	IaC for SageMaker and OpenWhisk	Automated job triggering and teardown using terraform destroy	Ensures no idle resource remains post-job; minimizes over-provisioning	↓ Runtime waste, ↑ dynamic provisioning
Glue Preprocessing	ETL pipeline integrated before inference	Glue Runtime = 3–5 mins for batch prep	Cleans and aligns data once per batch; avoids real-time prep per device	↓ Repeated preprocessing cost
Hybrid Invocation Control	Load-Aware Queueing for Kafka ↔ Dynamic Routing	Real-time jobs → OpenWhisk, High-latency tolerant jobs → SageMaker Batch	Maintains pipeline responsiveness without overloading either execution path	↑ Throughput, ↓ queue congestion