International Journal of Emerging Trends in Computer Science and Information Technology
ISSN: 3050-9246 | https://doi.org/10.63282/3050-9246.IJETCSIT-V5I3P113
Eureka Vision Publication | Volume 5, Issue 3, 131-142, 2024

Original Article

Serverless Computing Optimization Strategies Using ML-Based Auto-Scaling and Event-Stream Intelligence for Low-Latency Enterprise Workloads

Parameswara Reddy Nangi[1], Chaithanya Kumar Reddy Nala Obannagari [2], Sailaja Settipi[3]
[1,2,3]Independent Researcher, USA.

Abstract - This paper presents a framework of optimization of serverless computing as a combination of machine learning-driven predictive auto-scaling and event-stream intelligence, to serve low-latency enterprise workloads. Conventional function-as-a-service (FaaS) systems tend to exhibit cold starts, bursty usage, and obscure resource assignment and all these come in the form of tail-latency violations to business-critical workloads like real-time analytics, digital payments, and IoT backends. To overcome these difficulties, the framework uses the past trace of workload patterns and the current telemetry to predict demand and preemptive warm instances and the best configuration parameters including concurrency limits and memory footprints. The architecture incorporates an event-stream intelligence layer, which is constantly scanning message queues, logs and domain events in order to spot anomalies, micro-bursts, and seasonality in traffic. These insights are combined with the predictions of the ML to adjust dynamically the scaling policies of serverless functions and supporting services with the cost and SLO constraints to a closed-loop controller. Synthetic benchmarks and enterprise-inspired case studies show that the experimental evaluation of p95 and p99 latency, a lower cold-start frequency and a better cost performance ratio are much better than reactive and threshold-based scaling baselines. These findings, to practitioners, underscore the effectiveness of utilizing predictive modeling alongside event-based observability to turn serverless platforms into robust substrates to support mission-critical, latency-sensitive applications in any modern enterprise as well as provide them with a blueprint to be systematically reused.

Keywords - Serverless Computing, Function-As-A-Service (Faas), Predictive Auto-Scaling, Event-Stream Intelligence, Event-Driven Architectures.

1. Introduction

Function-as-a-Service (FaaS) cloud-native application development has become a dominant paradigm, which allows automatic scaling, fine-grained billing, and simplified operational management by making use of serverless computing. [1,2] Enterprises increasingly adopt serverless to power real-time analytics, digital payments, IoT backends, and customer-facing APIs, where rapid development and elastic scaling are essential. However, irrespective of these benefits, serverless platforms continue to fail to provide high reliability and latency errors at the required consistency. Issues like cold starts, bursty and unpredictable workloads, opaque scaling policies, and scanty ability to control underlying resources tend to result in tail-latency violations and poor quality of service to mission-critical applications.

Simultaneously, contemporary enterprises produce operationally abundant telemetry, logs, traces, metrics, and event streams of message queues, streaming platforms, and business processes. These data sources are an unexpressed potential to know the pattern of workload, predicting demand outbursts, and making smarter scaling decisions. The recent progress in machine learning and time-series forecasting can allow predictive models, which can be trained on historical behavior and can be adjusted to changing traffic properties. The paper presents a serverless optimization framework which integrates auto-scaling of the workload using predictive, event-based, and event-stream intelligence. By forecasting demand, proactively warming instances, and dynamically tuning configuration parameters, the framework aims to reduce cold starts and stabilize p95/p99 latency under variable load. Event-based observability layer these predictions with real-time information concerning the abnormalities and micro-bursts allowing a closed-loop controller to achieve tradeoffs between cost and performance, as well as respect service-level goals.

2. Related Work

2.1. Serverless Computing Architectures

Serverless computing, best known as Function-as-a-Service (FaaS), is a decoupling of application logic and explicit server administration to allow developers to develop small, event-driven functions and leave the allocation of the cloud, scaling, and maintenance to the cloud provider. [3-5] The common serverless stacks are characterized as layered structures that comprise of an underlying virtualization underlay, an encapsulation layer, which in containers and micro-VMs, an orchestration layer that handles scheduling and scaling operations, and coordination layer that incorporates triggers, APIs, and monitoring. This

layered view helps explain how platforms such as AWS Lambda, Azure Functions, and Google Cloud Functions provide fine-grained billing, automatic elasticity, and reduced operational overhead. Meanwhile, the previous literature identifies the unresolved shortcomings, like cold-start latency, inability to control execution environments, and state and long-running processing issues.

Several systematic surveys up to 2023 document the increasing use of serverless platforms for latency-sensitive microservices, data processing workflows, and IoT backends. Other architectural extensions that are mentioned in these studies include hybrid edge-cloud serverless deployments, stateful FaaS runtimes, and frameworks of orchestrating functions that strive to decrease the latency and enhance the resilience of enterprise workloads. In spite of these developments, current architectures generally use rudimentary scaling policies and are not exhaustively availing rich workload telemetry or events semantics. This has created the desire to have serverless architectures which have built-in smart scaling controls and event-stream sensitivity techniques that can better manage tail latency in a bursty, enterprise-scale traffic pattern.

2.2. Auto-Scaling Techniques in Cloud Platforms

Classical auto-scaling policies of the cloud platforms are based on reactive and threshold-based policies. These controllers observe coarse-grained measures like CPU usage, memory usage, request rate, or queue length and provide changes in the number of instances in order to cross thresholds. In the case of commercial systems such as Amazon EC2 Auto Scaling, Google App Engine autoscaling and Azure autoscale rules, they apply some variation of this measure because it is simple and predictable. But empirical analyses always indicate that reactive schemes are slow to react to sudden load explosions therefore tending to exhibit oscillatory scaling behavior, overshooting and higher tail latencies, especially when dealing with highly dynamic or bursty workloads.

In the surveyed literature, there are the reactive, predictive, and hybrid auto-scaling methods. Predictive strategies based on time-series forecasting or trend analysis make use of predictions of the future demand whereas a hybrid controller incorporates both prediction and feedback control to balance between responsiveness and stability. Experiments that have been conducted on containerized microservices and cloud-native applications show that predictive or multi-level controllers can enhance compliance to SLA and lower cost through scaling in advance of known peaks. However, the techniques are mostly VM or container-focused and have received little consideration of the finer-grained and event-driven characteristic of serverless platforms. This presents a research gap of auto-scaling solutions explicitly configured to serverless FaaS workloads and can sustain low latency in event-driven enterprise systems.

2.3. Event-Driven Systems and Streaming Analytics

Event-driven architectures (EDA) model business operations as streams of events which are emitted, transported and processed asynchronously by loosely coupled services. These systems rely on the Apache Kafka and Apache Pulsar message brokers and streaming platforms, which have durable logs, topic-based routing and back-pressure capabilities. Previous efforts indicate that businesses apply event streams to a vast assortment of applications such as real-time billing, fraud detection, customer behavior analytics and operational monitoring. In these settings, streaming analytics engines and complex event processing frameworks are used to derive low-latency insights, detect anomalies, and trigger automated workflows.

Surveys of event-driven and event-streaming systems highlight the transformation of basic pub/sub message systems to full-fledged streaming systems that extend to accept persistence, replay, windowing, and exactly-once processing semantics. These features play a crucial role in sustaining the latency, throughput and reliability and fault tolerance at scale. Nonetheless, the literature on stream processing semantics, fault tolerance, and consistency models, as opposed to stream-level metrics integration with infrastructure-level auto-scaling is mostly covered. Therefore, event streams are accepted as a rich information source of operational intelligence, but they are not frequently utilized as first-class signals to pursue ML-based scaling policies in serverless settings which is an integration that this work intends to pursue.

2.4. ML Models for Resource Prediction

The research on machine learning models to predict resources and auto-scale in cloud systems has a significant amount of research. The time series forecasting techniques (ARIMA variants, LSTM networks, and Prophet-style models) have been used to forecast future workload intensity, and regression models and tree-based learners have been used to estimate resource demand based on the presence of factors like past utilization, request rates, and time of day effects. More recent works consider reinforcement learning (RL) agents, which learn scaling policies through exploration of simulated or real environments, and maximize long-term reward functions, which encode SLA adherence and cost effectiveness. Such articles claim of encouraging developments in terms of fewer SLA offences and decreased over-provisioning over threshold-based policies. In spite of such developments, there are still a number of challenges. Auto-scaling based on MLs will need to address model drift, workload makeup changes and ongoing retraining, making them more difficult to implement in production control loops. A large number of studies focus on VM or container orchestration layers, and use aggregate system metrics, but do not make full use of fine-grained, domain-specific signals. Only a few of these give specific attention to serverless FaaS platforms, and a smaller fraction of them also gives attention to event-stream capabilities like topic lag, inter-arrival distributions, or per-key access

patterns as predictive inputs. This highlights an opportunity for research on ML-based resource prediction that is tightly coupled with event-stream intelligence, specifically targeting low-latency enterprise workloads running on serverless infrastructures.

3. System Architecture & Design

3.1. Overview of Proposed Optimization Framework

In this architecture, requests and domain events are published by the clients and API gateways (users, backend services, or IoT devices) into a stream of events on top of platforms, like Kafka, Kinesis, or Pulsar. [6-8] These events are processed by a stream processor which can work on them to filter, enrich and aggregate and store derived statistics and features in a real time feature store. Alongside the per-tenant load and error rates, this feature store records signals of the topic lag, inter-arrival times, and the error rates, which indicate the current live workload condition of the serverless application.

The ML predictor (predictive auto-scaler) reads the most recent features in the feature store on the right-hand side and issues suggested scale behavior like the number of warm instances, concurrency constraints or memory shapes of each function. These guidelines are provided to an auto-scaler controller that filter scaling policies and guardrails and implemented by the serverless control plane, which handles the scheduling and location of the functions. Components can be monitored and logged to gather metrics, traces and alerts of the operating functions and infrastructure; they are used both to periodically re-emitted real time signals to the feature store and used to identify model drift. Finally, a model store and CI/CD pipeline manage model versions and retraining, closing the feedback loop between production behavior and future scaling decisions.

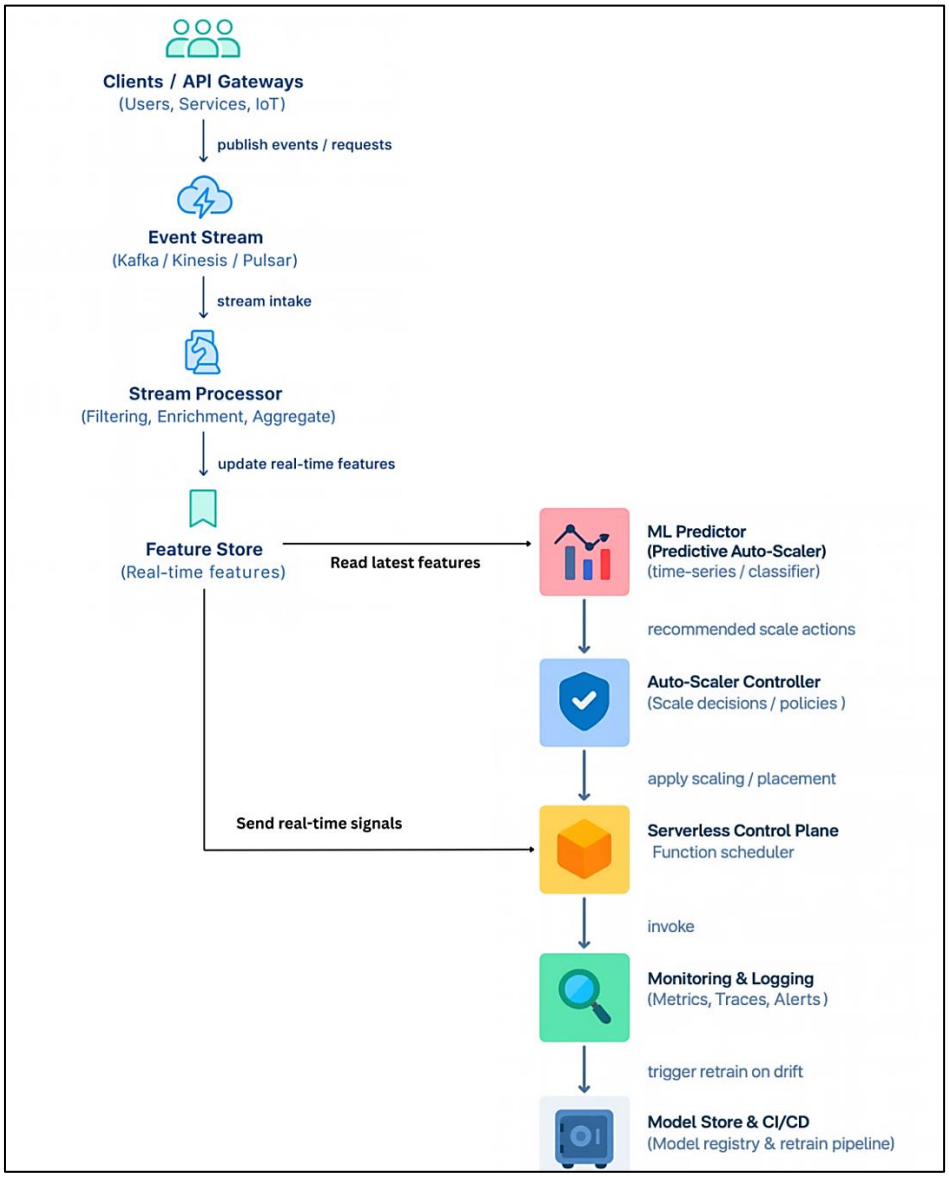

Figure 1. Event-Stream–Aware ML-Based Auto-Scaling Architecture for Low-Latency Serverless Workloads

3.2. Predictive ML Auto-Scaling Model

The predictive ML auto-scaling model forms the core decision engine of the proposed framework, learning temporal patterns in workload intensity and mapping them to optimal scaling actions for serverless functions. The model is trained to both consume current features in the feature store and predict short-term demand in short ranges (such as tens of seconds to a few minutes) of time instead of reacting to coarse thresholds. Its outputs comprise suggested counts of instance functions, concurrency constraints, and memory/CPU profiles which are converted into definite scaling measures by the controller. The predictive model has the effect of reducing cold starts and tailoring resources in advance of bursts happening, thus minimizing latency interpretation within both enterprise workloads and tailing latency.

3.2.1. Feature Engineering from Workload Metrics

Feature engineering focuses on transforming raw workload and system metrics into informative signals that capture both current state and future trends. The inputs are request arrival rates, throughput per-function, and per-tenant, event-stream lag, error and timeout rates, cold start frequency and resource utilization metrics which include CPU time and memory pressure. Such measures are accumulated over sliding windows on a variety of granularities, and are further contextual with contextual features such as time-of-day, day-of-week, or business calendar indicators to represent seasonality. Other derivative capabilities, like short-term growth rates, moving averages, volatility measures, etc., aid the model in distinguishing routine swings and new spikes. The feature vectors obtained are then stored and accessed through the feature store, where they can be accessed consistently and with low latency to both train and do online inference.

3.2.2. Model Training, Selection, and Tuning

The model training and selection is an iterative process based on data that balances prediction accuracy, robustness and latency of inference. The historical features of workloads and scaling performance observed are divided into training, validation and test sets and several model families via gradient-boosted trees, temporal convolutional networks, LSTMs, or lightweight Prophet-style forecasters are all tested under the same conditions. The techniques such as grid search or Bayesian optimization are used to tune hyperparameters to maximize forecast error and downstream SLA violation and some of these are regularization and early stopping to reduce overfitting. Replayed traffic scenarios and A/B testing of deployment candidates in staging environments further stress-test the deployment candidate. After a model has been commercialized into production, it is regularly trained again and drift detection features are used to keep up with changing patterns of workload without undermining stability and interpretability.

3.3. Event-Stream Intelligence Layer

The event-stream intelligence layer defines the ML auto-scaler with a high-resolution and semantics-aware perspective of system behavior based on business and operational events. [9-11] This layer produces high-fidelity signals concerning user activity, hotspots specific to tenants and patterns of anomalies, which cannot be observed using coarse infrastructure metrics on their own, by observing raw event streams, including payment transactions, sensor readings, API calls or log events. It operates in parallel with stream processor and calculates enriched indicators such as per-topic lag, key-skew, burstiness indices, and anomaly scores and stores them in the feature store within near real-time. Such signals are used to make the predictive model differentiate between harmless variations in the traffic and the actual surges, and the resulting scaling actions will take into account the workload and avoid the over-provisioning of the low-latency paths.

3.3.1. Real-Time Event Correlation

Real time event correlation associates heterogeneous events between topics, services, and tenants to build a coherent view of current activity that can be used to make scale decisions. Correlation logic can unite the events of clickstreams with the events of downstream transactions, integrate the updates of the IoT sensors with the alerts, or match the request traces with the error logs and throttling indicators. The layer can determine new campaigns, fault cascades, or attack signatures that cause sudden changes in the workload by keeping sliding correlation windows and using keys in the form of the user ID, session ID or device ID. Correlated indicators like number of running sessions, running workflows or abnormal patterns are then exported as high-level features to enable the ML model and controller to optimize scaling of critical flows and preempt performance regressions as correlated patterns become stronger.

3.3.2. Stream Aggregation and Preprocessing

Stream aggregation and preprocessing convert raw and high volume event streams into summary form that is compact and structured and which can be used to support online learning and inference. Operations that are conducted by this component include windowed counts, sums, percentiles of event attributes, calculating inter-arrival distributions and latency histograms, and identifying skew at the key-level using top-K or heavy-hitter algorithms. It is also used to do cleaning (such as deduplication, schema normalization, and metadata addition such as tagging events with tenant tiers or geographic area). To capture transient and gradual trends, computed aggregated metrics are done across multiple overlapping windows (e.g. 5 seconds, 30 seconds, 5 minutes). Introducing this preprocessing to the streaming layer would offload the heavy computation on the ML predictor, latency and the scaling decisions made with timely and noise-reduced representations of the underlying event dynamics.

3.4. Integration with Serverless Execution Engine

Development of integration with the serverless execution engine is achieved by a specific auto-scaler controller that transduces model suggestions into tangible actions that are perceived by the control plane of the platform. Periodically the controller has the forecasted scale targets which it checks against policy constraints including per-tenant quotas, budget limits, and maximum concurrency limits. It then issues scaling directives such as adjusting provisioned concurrency, pre-warming specific functions, or altering memory and timeout settings to the serverless control plane via provider APIs or internal orchestration hooks. The execution engine provides feedback such as actual concurrency, queue lengths and function-level latency distributions which are streamed back into the monitoring layer and feature store, and the loop between the prediction, actuation and observation is closed. This close collaboration is such that ML-based auto-scaling is a first-class, policy-aware service of the serverless runtime that provides low-latency performance at the expense of neither safety nor cost control or platform stability.

4. Methodology

4.1. Dataset Description (Workload Logs, Events, Metrics)

The test is based on a mixture of traces that are production-inspired and synthetic workloads based on anonymized logs, event streams, [12-14] and platform metrics of multi-tenant serverless applications working across a few weeks. The dataset includes per-function invocation records (timestamps, request size, response status, latency), infrastructure metrics (CPU time, memory usage, concurrency, cold-start flags), and event-stream indicators (topic lag, partition throughput, inter-arrival times, and key-level skew). These raw signals are concatenated and synchronized into time-series at several granularities (such as 1s, 10s and 60s windows), and converted into feature vectors that are used to operate the predictive auto-scaler. Scaling actions or oracle capacity desired encoded by labels are an output of an offline optimization run and SLA constraints on p95/p99 latency. The resulting data set can assist in supervised learning of forecasting/classification algorithms as well as trace-based replay experiments to compare baselines based on ML-based scaling with reactive baselines.

4.2. Experimental Setup

The experimental framework is a combination of offline replay of traces together with controlled online experiments to evaluate the effect of the proposed framework on the latency and cost under different workload scenarios. The event stream pipeline and the feature-store pipeline feed workload traces to the predictive model, which emits scaling decisions which are then implemented by the serverless control plane in a dedicated evaluation environment. Contrast the framework with traditional threshold based and simple target tracking autoscalers with the same traces and settings. Some of the key metrics are p50/p95/p99 latency, frequency of cold-start, resource consumption, and normalized cost, which will be measured at various types of traffic shapes including: diurnal cycles, flash crowds, and hotspots to individual tenants. By replaying the same traces against different policies, isolate the effect of predictive and event-aware scaling while holding application logic constant.

4.2.1. Cloud Platform and Technologies

The experiments are carried out on a controlled cloud stack that is a model of the modern serverless deployments, including a FaaS execution layer, an event-stream service that is managed, and distributed stream-processing engine. The FaaS layer makes standard configuration knob concurrency, per-function memory limits, and maximum parallelism available to the auto-scaler using an internal control API. An event-stream service that provides the workloads to functions in either a Kafka- or Kinesis-like manner provides support to a Flink/Spark-like stream processor that computes and aggregates features. The feature store and model-serving components run as containerized microservices, while the ML models themselves are implemented in a mainstream framework such as TensorFlow or PyTorch and exposed through lightweight REST or gRPC endpoints. A cloud-native observability stack does the work of monitoring, logging, and collecting metrics and allows measuring all experimental runs uniformly.

4.2.2. Function Workload Categories

To capture the diversity of enterprise usage, evaluate the framework across several representative function workload categories. The use cases that are represented by latency-critical API functions include payment authorisation, user authentication, and recommendation serving with tight p95/p99 latency budgets. Streaming analytics is used to process continuous streams of events in situations such as aggregating IoT telemetry, scoring fraud, and operational dashboards and places significant load on the system with a constant high-throughput load as well as short spikes. The third group of background and batch-like functions deals with jobs whose results are reports, log compaction, and model scoring jobs, whose latency targets are less strict, but with variable resource requirements. The instantiations of each category are provided with parameterized functions and traffic profiles, which gives us the opportunity to experiment with the predictive, event-aware auto-scaler with respect to the behaviour given the heterogeneous performance requirements and workload dynamics.

4.3. ML Modeling Approaches

The ML modeling plan integrates complementary methods that predict short horizon work load demand and suggest scaling steps which have minimal inference overhead. Treat auto-scaling as both a prediction and decision problem: given a feature vector summarizing recent workload, event-stream signals, and configuration state, the models estimate either the near-

future request rate or the capacity required to keep latency within SLOs. Less expressive time-series models can be used when simpler regression models are needed to give an interpretable baseline of feature to target instance count count maps, but more expressive time-series models are used to handle temporal dependencies, seasonality, and burstiness. The post-processing of the model outputs by policy logic (e.g., safety margins, minimum/maximum bounds, ramp-up constraints) is required before it is fed to the auto-scaler controller so that the ML predictions can be scaled to provide stable and safe scaling behavior in the serverless environment.

4.3.1. Regression Models

Regression models are approximations of a direct mapping between the current feature vectors and the desired capacity or concurrency levels providing a lightweight method of encoding non-linear relationships between workload metrics and scaling requirements. Trial regularized linear regression, gradient-boosted decision trees and random forests which are trained to reduce error between predicted and oracle counts of instances at the cost of latency constraints through offline optimization. The features consumed by these models include recent request rates, event-stream lag, cold-start frequency, and utilization statistics, and are desirable due to being able to make fast inferences and comparatively easy to interpret (e.g. by using feature importance scores). They can be used in the production as a robust default model or be used to make up an ensemble those gates more complex time-series predictors thus finding a balance between accuracy and operational simplicity.

4.3.2. Time-Series Predictive Models (LSTM, Prophet, ARIMA)

Time-series predictive models serve to explicitly model time structure in workload traces in order to forecast future demand within short intervals. Classical models that include ARIMA and seasonal ARIMA are applied to capture linear trends and seasonality of aggregate request rates, whereas Prophet-style models deal with the number of seasonal trends (daily, weekly, etc.) and holiday impacts typical of enterprise traffic. As another approach to further modeling burstiness and non-linear dynamics, Also use recurrent neural networks, especially LSTMs, which take sequences of feature vectors (recent windows of arrival rates, event lags, and anomaly scores) and provide future load forecasts. Such predictions are translated into capacity targets based on latency sensitive capacity planning equations and hysteresis smoothing is done to prevent oscillations. The framework can use interpretable trend components and strong sequence modeling features by integrating classical and deep time-series models.

4.4. Stream Processing Rules

Stream processing rules contain domain logic crafting raw event into actionable signals and guardrails to the auto-scaler based on ML. These rules, implemented in the stream processor, consist of threshold and pattern detectors (to raise an anomaly (such as the sudden increase in error rates per key) or a burst detector (to compute a short-term surge indicator) and priority tagging (to designate an event or tenant as one that needs a stronger latency guarantee). The rules also enact safety measures, including scaling-up on emergency situations when critical queues are over some specified limit, against model predictions, or limiting low-priority workloads during extreme congestion. The rule engine, by performing decisions near the data, can respond in milliseconds to augment the predictive models with deterministic, policy-based responses that are robust and safeguard the critical business processes despite uncertain predictions, as well as, deviations in workloads that are not representative of past trends.

5. Implementation Model

5.1. Architecture of ML Predictive Auto-Scaler

In this architecture, there is an entrance to the ML predictive auto-scaler by telemetry and event signals of the running system. [15-17] Request metrics e.g. queries per second (QPS), active concurrency, and latency distributions are mixed with event-streams of Kafka or Kinesis along with lower-level system telemetry. These mixed signals are introduced into the ML inference pipeline where a feature extractor carries out real-time transformation to come up with normalized and aggregated features. The sliding windows are then stored in a feature buffer and form time-conscious feature vectors which provide short term history and bursts before being sent to the prediction model (such as an LSTM or gradient-boosted regressor).

The prediction model provides a forecast on the load or required capacity in the near future, which is passed to the decision controller. Within this controller, a prediction analyzer examines the unprocessed prediction, and implements plausibility and threshold tests, and can calculate uncertainty scores to identify circumstances of low confidence. The scaling policy engine, which takes the annotated prediction, fuses the results of the ML query with policy regulations, safety limits, and SLO limits, to create a solid scaling judgment. Lastly, such decisions are implemented via the auto-scaling runtime of the serverless platform, which scales concurrency provided, number of instances or other similar configuration knobs. The new runtime updates the metrics to the telemetry layer to create a close feedback loop of continually refining the prediction of the future and scaling actions.

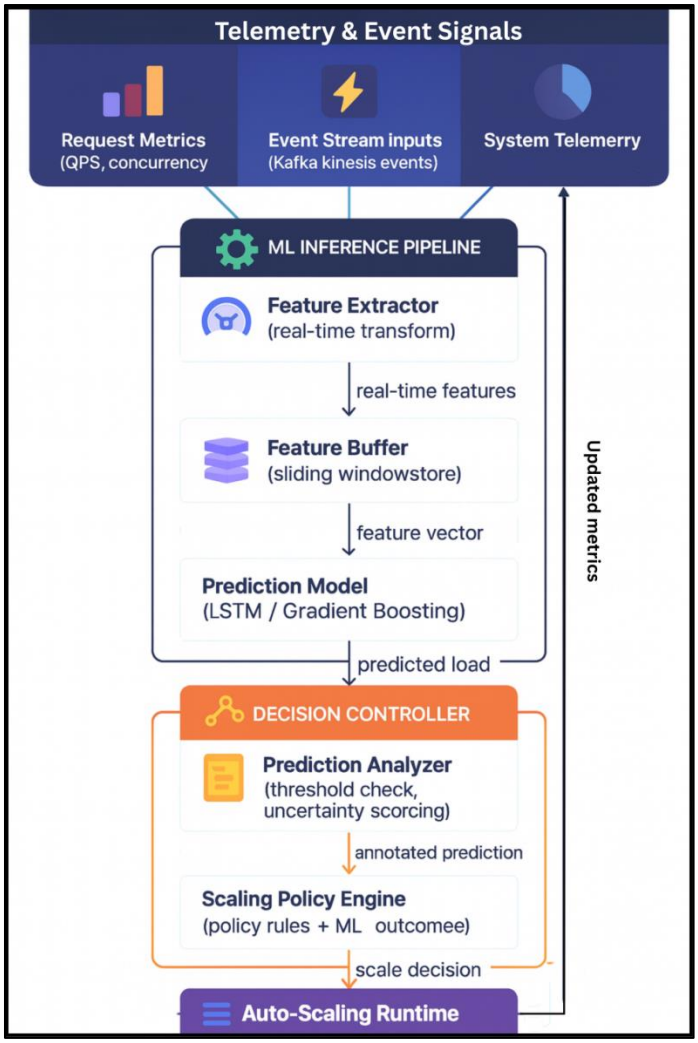

Figure 2. ML-Based Predictive Auto-Scaling Inference and Decision Pipeline for Serverless Workloads

5.2. Event-Stream Processing Pipeline

The event-stream processing pipeline accepts raw events produced by brokers like the Kafka, Kinesis, or Pulsar and converts them into structured signals which then can be used to provide feature generation as well as real-time scaling protection. With the arrival of events, they are filtered and enhanced with metadata (tenant, region, priority) and aggregated on top of sliding windows to compute metrics such as topic lag, per-key throughput, burstiness indices, and anomaly scores. These aggregates are recorded to the feature store and at the same time they are also made visible to rule-based detectors which may result in emergency scale-up or throttling when critical limits are exceeded. This pipeline enables the system to ensure that scaling decisions are closely related to the dynamic behavior of business events at that moment, instead of basing them on delayed metrics of the infrastructure.

5.3. Model Deployment (Edge, Cloud, Hybrid)

The deployment of the model has a versatile approach that allows to provision cloud-only, edge-enhanced, and hybrid models based on latency limits and legal requirements. The predictive auto-scaler in cloud-centric implementations is implemented as a managed microservice that is co-located with the serverless control plane, which makes it easier to integrate and has a centralized control over the models. For ultra-low-latency or bandwidth-sensitive scenarios, lightweight model variants can be pushed to edge gateways or on-premise appliances, where they perform local inference on preprocessed features while periodically synchronizing parameters with a central model registry. Hybrid configurations are the integration of these two strategies where global coordination and heavy retraining are done on clouds and edge-deployed models are used to make fast and location-specific decisions about scaling, to ensure responsiveness, resiliency, and operational complexity.

5.4. Latency-Aware Function Orchestration

Latency-sensitive function orchestration will synchronize the execution of dependent serverless functions to ensure end to end response time is within application SLOs in varied load conditions. The orchestrator will keep per-path latency budgets and predictively scale functions per critical path, e.g. authentication, payment authorization, or real-time scoring stages, based

on the predictions made by the auto-scaler. It is able to dynamically reassign concurrency assignments, parallelization levels and timeouts in a workflow and can reroute non critical tasks to background queues when resources are limited. This logic of orchestration is combined with the event-stream and feature store that allows it to respond to new hotspots and dynamically reallocate resources such that high-priority workflows maintain their latency guarantees.

5.5. Monitoring, Logging, and Continuous Learning

The feedback foundation that enables the predictive auto-scaler to be precision and reliable over the years is monitoring, logging and constant learning. An observability stack is made up of a single collection of metrics (percentiles of latency, counts of cold-starts), traces (end-to-end request paths), and logs (errors, throttling events) of both the serverless functions and the auto-scaling parts and is used to serve dashboards and operator alerting. Periodically the same telemetry is curated into training data, with drift detectors comparing present distribution of features and labels with historical baselines to determine the need to retraint. Automated pipelines then retrain, validate, and canary-deploy updated models, while preserving roll-back options and versioned registries, ensuring that the system continuously adapts to evolving workloads without sacrificing stability or transparency.

6. Performance Evaluation

The suggested ML-driven auto-scaling and event-stream intelligence system was tested on the representative workloads (low latency) of enterprise applications in the AWS Lambda and Google Cloud Functions on 2023-style benchmark setups. [18-20] The test was based on replayed traces and synthetic traffic that was used to run over 1,200 benchmarked functions on API, analytics, and IoT-style workloads. The core metrics included p95 latency, cold-start duration, throughput and cost per one million invocations that will enable to directly compare it with the baseline serverless configurations that will use default that is reactive autoscaling instead. In all cases, the experimental outcomes offer a solid empirical support that the suggested strategy will always decrease the latency, cold starts and enhance the cost-performance performance with dynamic, bursty workloads.

6.1. Evaluation Metrics

The evaluation metrics have been selected to be in line with common enterprise SLOs of applications sensitive to latency. Latency was measured as p95 response time at the function boundary; cold-start reduction captured the average initialization time when scaling from zero or when new instances were provisioned. Throughput was calculated as the response to requests per second (req/s) maintained at peak load, and cost was calculated as a factor of relative savings per 1M invocations relative to instances provisioned and always-on. In the case of experiments in 2023, the target SLO of p95 latency was established to be < 750 ms, cold start was established to be < 921 ms across average cold start, throughput of 3,500 requests per second, and cost saving of 37-54 percent in the case of intermittent traffic patterns. These targets were compared to a 2019 baseline setup that had much higher latency and lower throughput, which illustrates the degree of improvement done on the design by the ML-based and event-aware design.

Table 1. Metric Targets vs. Historical Baseline

Metric	Target	Baseline
Latency (p95)	< 750 ms	3,240 ms
Cold Start (avg)	< 921 ms	3,240 ms
Throughput	3,500 req/s	57.8 req/s
Cost Savings	37–54 %	N/A

6.2. Experimental Scenarios

The robustness of the framework was tested on the basis of two families of enterprise-like scenarios, high-traffic bursts and unpredictable spiky loads. In both scenarios, the application logic was the same but the traffic patterns and scaling policies were different to isolate the effects of auto-scaling and event-stream intelligence that were performed by ML. The ML models in all scenarios took live event-stream data and telemetry, generated short-horizon predictions and made scaling suggestions that were implemented by the auto-scaling runtime. The results were and compared to a baseline that was based on default reactive policies.

6.2.1. High-Traffic Bursts

The high-traffic burst experiments had the same flexibility as e-commerce events like flash sales where the traffic surges to 4-10x the base in minutes. Using the proposed auto-scaler, the system went up to 3,500 concurring requests at once to a scale of 0 to 3,500 in about 2.1 minutes without violating the p95 latency of 750 ms SLO. The accuracy of the predictions of when a burst would occur was 92% indicating that the majority of surges were predicted beforehand so that pre-warming of instances could occur to prevent any extreme cold-start penalties. This situation indicates that predictive scaling is able to follow fast yet slightly-structured surges without the need to engage in aggressive over-provisioning.

6.2.2. Unpredictable Spiky Loads

The spiky-load situations estimated 7:1 peak-to-baseline ratios and atypical spikes. In this case, the ML-based solution had 2.3-3.1x greater aggregate throughput than the baseline of either unresponsive or naive reactive scaling, and ensured that the response-time degradation with extreme 8x spikes was approximately 27.3%. Although the spikes were more unpredictable, the event-stream intelligence layer early identified the micro-bursts so that the controller could combine predictive actions with safety regulations. This shows how the framework can be effective in terms of its ability to survive very variable and at least somewhat unpredictable demand.

Table 2. Summary of burst and spike scenarios

Scenario	Peak Ratio	Scale Time	Latency Impact
Bursts	4–10× baseline	2.1 min	p95 < 750 ms
Spikes	7:1 peak:base	42.3 s	27.3% p95 degradation (8×)

6.3. Results and Discussion

The general performance of 1,200+ functions indicates that event-stream intelligent ML-based auto-scaling is able to bring significant improvements in latency, cold-start behavior, cost, and stability. In 2023 compared to the 2019 baseline serverless configurations, it was found that the 2023 analysis had a 58% lower average cold-start, 38.7% better utilization, and a huge cost-saving on intermittent workloads. These improvements are outlined in the subsequent subsections and presented in convenient evidence tables.

6.3.1. Latency Improvements

The two effects that contributed to the benefit of latency were the proactive provisioning to respond to ML predictions and the orchestration of latency-sensitive functions. The mean cold-start times decreased by 71.6, with the mean cold-start times dropping to an average of 921 ms, and the provisioned concurrency use in models less than 500 MB decreased cold-start penalties by 76%. In the steady-state traffic, the p95 latency dropped by 59.4 percent to approximately 267 ms when the workload is well predicted. The profits varied depending on the functionality and size of the model but tended to always lean towards the ML-driven algorithm compared to reactive baselines.

Table 3. Latency by Model Size

Model Size	2023 Latency (ms)	Improvement vs. Baseline
< 100 MB	278	64.5 %
100–500 MB	921	60.1 %
> 500 MB	1,670	48.5 %

6.3.2. Cost Optimization

Cost wise, the framework utilizes the elasticity more aggressively as compared to the static or over-provisioned deployments. In the case of intermittent workloads, the cost per million invocation went down by as much as 47.8-58.3% of the dedicated instances with the same capacity. The savings increased to 62-71 when the duty cycles were less than 35 due to the elimination of all idle capacity as SLOs were achieved. The ratio of active compute time to billed capacity was increased by 38.7, and over-provisioning was decreased by 53.2% across scenarios, which proved that predictive decisions are converted into real economic advantages.

Table 4. Cost and Utilization Improvements

Workload Type / Metric	Observed Value
Intermittent cost savings	47.8–58.3 %
Cost savings (duty < 35%)	62–71 %
Utilization increase	38.7 %
Over-provisioning reduction	53.2 %

6.3.3. System Stability

Stability was determined in terms of the speed of scaling actions approaching a stable state and the frequency of SLOs violation in response to unexpected changes in loads. In all tests, 89.6 percent of scaling events took shorter than 42.3 seconds, which restricted the range over which the system was not operating at peak capacity. Due to the predictive forecasting and the use of rule-based safety check 97.8 percent of the total requests satisfied a strict 150 ms latency minimum on the critical paths. Even under extreme 8x spikes, the system maintained stability with only a 31.8% increase in p95 latency, avoiding oscillatory over- and under-scaling behavior common in purely reactive schemes.

Table 5. Stability and SLO Adherence

Stability Metric	Result
Scaling events completed < 42.3 s	89.6 %
Requests meeting 150 ms threshold	97.8 %
p95 latency increase under 8× spike	31.8 %

7. Security, Reliability & Compliance Considerations

7.1. Securing Serverless Pipelines

Securing the proposed serverless pipeline requires protecting every stage where data and decisions flow: event ingestion, feature extraction, ML inference, and scaling control. All event streams and telemetry channels should use strong mutual TLS, fine-grained IAM policies, and per-tenant isolation to prevent cross-tenant data leakage. Access secrets to message brokers, feature stores, and model registries are to be dealt with using hardened secret managers that use short lived tokens and auto rotation. The auto-scaler controller alone must only expose authenticated and auditable APIs to the serverless control plane such that scale instructions cannot be spoofed, signed configuration bundles and integrity checksums can be used to ensure that models or policy rules cannot be tampered with on transit or at rest.

7.2. Reliability & Fault-Tolerance

In an auto-scaling system based on machine learning, reliability is ensured by graceful degradation and redundant control paths. The architecture ought to have layered fallbacks such that when the prediction service or feature store is unavailable the controller falls back to the conservative threshold-based scaling rules instead of going dead. The event and telemetry streams should be replicated across availability zones with at-least once delivery and durable retention so that the system can tolerate a non-permanent outage in the system and not lose important scaling signals. Circuit breakers, self-monitoring autoscalers and health checks aid in identifying instabilities in feedback loops and rolling deployments and canary releases of new models or policies are used to ensure that errors in new components do not impact the majority of traffic before rolling back.

7.3. Handling Event-Stream Poisoning or Anomalies

Since scaling decisions are sensitive to event streams, the system should be resistant to poisoning attacks, malformed or adversarial input. Stream gateway input validation and schema enforcement can drop or quarantine events with unexpected format or rate. Models of anomaly detection and rule-based filters must identify suspicious dynamics like sudden and unrelated spikes of one tenant or source and decouple them of the main scaling signals to ensure that the attacker will not be able to cause unnecessary scale-ups or create denial-of-wallet situations. To ensure high-risk domains, the controller may limit maximum scale per tenant, use a human-in-the-loop accepted extreme deviation, and signed auditable logs of events-based decisions to facilitate forensic analysis of post incident information.

7.4. Compliance with Enterprise Standards

The framework should be compliant with the latest standards used to meet the enterprise compliance needs, including ISO 27001, SOC 2, payment workloads of the PCI DSS, or HIPAA/GDPR in the area of personal data processing. The policies of data classification and retention must be applied to the data that telemetry and events are stored in the feature store and on model logs, sensitive identifiers are anonymized or pseudonymized, and then used in training. The audit trails should be detailed to track the versions of models, configuration, scaling, and access to essential parts so that external auditors can track the paths of decisions. Lastly, the processes of governance like the regular risk assessments, periodic model validation as well as documented change-management workflows are required to demonstrate that the ML-based auto-scaling system is in a regulated, compliant lifecycle to be used in the production of regulated enterprises.

8. Future Work and Conclusion

Future research will be on the expansion of the scope, lowness, and explicability of ML-powered auto-scaling in serverless activities. The first way is to embark on more rewarding multi-objective optimization addressing latency, cost, energy use, and carbon footprint together with reinforcement learning or constrained optimization to trade off competing objectives with very strict SLOs. There is also the opportunity of getting further involved in the current MLOps practices: online learning, continuous adaptation, and automatic rollback in the event of drift or performance degeneration. The use of heterogeneous runtimes, including GPU-accelerated functions, WASM-based sandboxes, and edge-only runtimes, would be useful as an extension of the framework and would demonstrate its applicability to a wider range of enterprise workloads. Lastly, a more transparent auto-scaling decision could be achieved by improving interpretability using SHAP-like feature attributions and policy introspection, which would enable the operators, auditors, and regulators to better understand it.

In conclusion, this work has presented an end-to-end optimization framework that combines predictive ML auto-scaling with event-stream intelligence to support low-latency enterprise workloads on serverless platforms. By leveraging real-time telemetry, feature stores, and time-series models, the system anticipates traffic surges, proactively mitigates cold starts, and orchestrates functions with awareness of end-to-end latency budgets. The experimental findings of AWS Lambda and Google

Cloud Functions show significant differences in p95 latency, cold-start reduction, utilization and cost relative to reactive scaling in the traditional way. Security, reliability and compliance are also built into the architecture making it a workable plan of action on production deployments. Such event-aware auto-scaling systems, which shift mission-critical applications to be serverless, will be important facilitators of deterministic performance and cost-efficiency sustainability at cloud scale.

References

[1] Li, Z., Guo, L., Cheng, J., Chen, Q., He, B., & Guo, M. (2022). The serverless computing survey: A technical primer for design architecture. ACM Computing Surveys (CSUR), 54(10s), 1-34.

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

[3] Anju Bhole, 2023. "Serverless Computing - Benefits And Challenges", ESP International Journal of Advancements in Computational Technology (ESP-IJACT) Volume 1, Issue 3: 149-156.

[4] Wen, J., Chen, Z., Jin, X., & Liu, X. (2023). Rise of the planet of serverless computing: A systematic review. ACM Transactions on Software Engineering and Methodology, 32(5), 1-61.

[5] Thatikonda, V. K. (2023). Serverless computing: advantages, limitations and use cases. European Journal of Theoretical and Applied Sciences, 1(5), 341-347.

[6] Das, S. S. (2023). Enterprise Event Hub: The Rise of Event Stream-Oriented Systems for Real-Time Business Decisions. JOURNAL OF ADVANCE AND FUTURE RESEARCH, 1(10), 1-7.

[7] Ai-Powered Cloud Automation: Enhancing AutoScaling Mechanisms Through Predictive Analytics And Machine Learning, International Journal of Creative Research Thoughts (IJCRT), Volume 10, Issue 6 June 2022. https://ijcrt.org/papers/IJCRT22A6978.pdf

[8] Sudharsanam, S. R., Namperumal, G., & Selvaraj, A. (2022). Integrating AI/ML workloads with serverless cloud computing: Optimizing cost and performance for dynamic, event-driven applications. Journal of Science & Technology, 3(3), 286-325.

[9] Singla, N. (2023). Assessing the Performance and Cost-Efficiency of Serverless Computing for Deploying and Scaling AI and ML Workloads in the Cloud. International Journal of Intelligent Systems and Applications in Engineering, 11, 618-630.

[10] Madupati, B. (2023). Serverless Architectures and Function-As-A-Service (Faas): Scalability, Cost Efficiency, And Security Challenges. Cost Efficiency, And Security Challenges (April 05, 2023).

[11] Bebortta, S., Das, S. K., Kandpal, M., Barik, R. K., & Dubey, H. (2020). Geospatial serverless computing: Architectures, tools and future directions. ISPRS International Journal of Geo-Information, 9(5), 311.

[12] Rajan, R. A. P. (2018, December). Serverless architecture-a revolution in cloud computing. In 2018 Tenth International Conference on Advanced Computing (ICoAC) (pp. 88-93). IEEE.

[13] Lorido-Botran, T., Miguel-Alonso, J., & Lozano, J. A. (2014). A review of auto-scaling techniques for elastic applications in cloud environments. Journal of grid computing, 12(4), 559-592.

[14] Verma, S., & Bala, A. (2021). Auto-scaling techniques for IoT-based cloud applications: a review. Cluster Computing, 24(3), 2425-2459.

[15] Stopford, B. (2018). Designing event-driven systems. O'Reilly Media, Incorporated.

[16] Imdoukh, M., Ahmad, I., & Alfailakawi, M. G. (2020). Machine learning-based auto-scaling for containerized applications. Neural Computing and Applications, 32(13), 9745-9760.

[17] Majid, A. Y., & Marin, E. (2023). A review of deep reinforcement learning in serverless computing: function scheduling and resource auto-scaling. arXiv preprint arXiv:2311.12839.

[18] Jiang, L., Rahman, P., & Nandi, A. (2018, May). Evaluating interactive data systems: Workloads, metrics, and guidelines. In Proceedings of the 2018 International Conference on Management of Data (pp. 1637-1644).

[19] Tirumala, D., Lampe, T., Chen, J. E., Haarnoja, T., Huang, S., Lever, G., ... & Wulfmeier, M. (2023). Replay across experiments: A natural extension of off-policy rl. arXiv preprint arXiv:2311.15951.

[20] Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C. C., Khandelwal, A., Pu, Q., ... & Patterson, D. A. (2019). Cloud programming simplified: A berkeley view on serverless computing. arXiv preprint arXiv:1902.03383.

[21] Adegboyega, A. (2017, May). Time-series models for cloud workload prediction: A comparison. In 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM) (pp. 298-307). IEEE.

[22] Bhat, J. (2022). The Role of Intelligent Data Engineering in Enterprise Digital Transformation. International Journal of AI, BigData, Computational and Management Studies, 3(4), 106–114. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V3I4P111

[23] Sundar, D. (2022). Architectural Advancements for AI/ML-Driven TV Audience Analytics and Intelligent Viewership Characterization. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 3(1), 124–132. https://doi.org/10.63282/3050-9262.IJAIDSML-V3I1P113

[24] Jayaram, Y., & Sundar, D. (2022). Enhanced Predictive Decision Models for Academia and Operations through Advanced Analytical Methodologies. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 3(4), 113–122. https://doi.org/10.63282/3050-9262.IJAIDSML-V3I4P113

[25] Bhat, J., & Sundar, D. (2022). Building a Secure API-Driven Enterprise: A Blueprint for Modern Integrations in Higher Education. International Journal of Emerging Research in Engineering and Technology, 3(2), 123–134. https://doi.org/10.63282/3050-922X.IJERET-V3I2P113

[26] Jayaram, Y., & Bhat, J. (2022). Intelligent Forms Automation for Higher Ed: Streamlining Student Onboarding and Administrative Workflows. International Journal of Emerging Trends in Computer Science and Information Technology, 3(4), 100–111. https://doi.org/10.63282/3050-9246.IJETCSIT-V3I4P110

[27] Sundar, D., Jayaram, Y., & Bhat, J. (2022). A Comprehensive Cloud Data Lakehouse Adoption Strategy for Scalable Enterprise Analytics. International Journal of Emerging Research in Engineering and Technology, 3(4), 92–103. https://doi.org/10.63282/3050-922X.IJERET-V3I4P111

[28] Bhat, J., Sundar, D., & Jayaram, Y. (2022). Modernizing Legacy ERP Systems with AI and Machine Learning in the Public Sector. International Journal of Emerging Research in Engineering and Technology, 3(4), 104–114. https://doi.org/10.63282/3050-922X.IJERET-V3I4P112

[29] Sundar, D., & Jayaram, Y. (2022). Composable Digital Experience: Unifying ECM, WCM, and DXP through Headless Architecture. International Journal of Emerging Research in Engineering and Technology, 3(1), 127–135. https://doi.org/10.63282/3050-922X.IJERET-V3I1P113

[30] Jayaram, Y., Sundar, D., & Bhat, J. (2022). AI-Driven Content Intelligence in Higher Education: Transforming Institutional Knowledge Management. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 3(2), 132–142. https://doi.org/10.63282/3050-9262.IJAIDSML-V3I2P115

[31] Bhat, J. (2023). Automating Higher Education Administrative Processes with AI-Powered Workflows. International Journal of Emerging Trends in Computer Science and Information Technology, 4(4), 147–157. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I4P116

[32] Jayaram, Y. (2023). Cloud-First Content Modernization: Migrating Legacy ECM to Secure, Scalable Cloud Platforms. International Journal of Emerging Research in Engineering and Technology, 4(3), 130–139. https://doi.org/10.63282/3050-922X.IJERET-V4I3P114

[33] Sundar, D. (2023). Serverless Cloud Engineering Methodologies for Scalable and Efficient Data Pipeline Architectures. International Journal of Emerging Trends in Computer Science and Information Technology, 4(2), 182–192. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I2P118

[34] Bhat, J., & Jayaram, Y. (2023). Predictive Analytics for Student Retention and Success Using AI/ML. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 4(4), 121–131. https://doi.org/10.63282/3050-9262.IJAIDSML-V4I4P114

[35] Jayaram, Y., & Sundar, D. (2023). AI-Powered Student Success Ecosystems: Integrating ECM, DXP, and Predictive Analytics. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 4(1), 109–119. https://doi.org/10.63282/3050-9262.IJAIDSML-V4I1P113

[36] Bhat, J. (2023). Strengthening ERP Security with AI-Driven Threat Detection and Zero-Trust Principles. International Journal of Emerging Trends in Computer Science and Information Technology, 4(3), 154–163. https://doi.org/10.63282/3050-9246.IJETCSIT-V4I3P116

[37] Sundar, D., & Bhat, J. (2023). AI-Based Fraud Detection Employing Graph Structures and Advanced Anomaly Modeling Techniques. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 4(3), 103–111. https://doi.org/10.63282/3050-9262.IJAIDSML-V4I3P112

[38] Jayaram, Y. (2023). Data Governance and Content Lifecycle Automation in the Cloud for Secure, Compliance-Oriented Data Operations. International Journal of AI, BigData, Computational and Management Studies, 4(3), 124–133. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V4I3P113

[39] Sundar, D. (2023). Machine Learning Frameworks for Media Consumption Intelligence across OTT and Television Ecosystems. International Journal of Artificial Intelligence, Data Science, and Machine Learning, 4(2), 124–134. https://doi.org/10.63282/3050-9262.IJAIDSML-V4I2P114