International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 10 | Oct - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

Image Auto-Compression using Sharp and AWS Lambda

Ms. Farhina S. Sayyad
Dept. Of Computer Engineering
D. Y. Patil College Of Engineering
Savitribai Phule Pune University,
Pune,India
fssayyad@dypcoeakurdi.ac.in

Aishwarya Shirgavi
Dept. Of Computer Engineering
D. Y. Patil College Of Engineering
Savitribai Phule Pune University,
Pune,India
aishushirgave98@gmail.com

***Abstract*— In today's digital era, users frequently upload high-resolution images, which often lead to system performance issues, slower load times, and excessive cloud storage usage. Manual image optimization remains inefficient and prone to human error for both developers and end-users. This paper introduces an automated, serverless image optimization pipeline utilizing AWS Lambda in combination with the Sharp.js library. When an image is uploaded to Amazon S3, it activates a Lambda function that automatically compresses and optimizes the image into a web-friendly format without noticeable quality degradation. This approach enables real-time image compression without the need for backend server management, thereby minimizing storage requirements, improving application speed, and enhancing user experiences across various platforms.**

In the modern internet-driven landscape, images represent a significant portion of the data transmitted across both web and mobile applications. Studies indicate that over 65% of webpage data weight is attributed to images, underlining the necessity of efficient image management. While high-resolution visuals are crucial for superior user engagement, they increase bandwidth consumption, load time, and cloud storage expenses. Traditional optimization approaches demand manual pre-processing or rely on specialized backend servers, which introduces inefficiency, cost, and maintenance challenges.

This study proposes a completely automated, serverless pipeline for image compression and optimization using AWS Lambda and Sharp.js. Leveraging AWS Lambda's eventdriven framework, the system triggers compression operations whenever new images are uploaded to S3. Sharp.js, built upon the efficient libvips engine, performs resizing and compression operations while maintaining visual quality. The integration of serverless computing with this high-performance library ensures real-time automation, scalability, and cost efficiency. Furthermore, this research introduces two innovative enhancements:

1. **A Deep Reinforcement Learning (DRL)-based predictive resource provisioning mechanism that mitigates cold start latency.**
2. **A Semantic-Aware Adaptive Compression (S-ADC) algorithm that intelligently modifies compression settings based on image content and semantic complexity.**

Experimental evaluations conducted across formats such as JPEG, PNG, WebP, and AVIF reveal considerable reductions in file size while preserving visual fidelity. The proposed system not only enhances accessibility for users with limited bandwidth but also reduces cloud expenses and supports sustainable computing practices. By merging serverless infrastructure with adaptive intelligence, this work delivers a scalable, cost-effective, and eco-friendly solution for image optimization applicable to real-world web and mobile platforms.

Keywords*—Cloud Computing, Serverless Architecture, AWS Lambda, Sharp.js, Image Compression, Reinforcement Learning, Adaptive Compression, Media Optimization, Cloud Efficiency.

I. INTRODUCTION

Modern digital platforms such as Instagram, Facebook, and various e-commerce applications rely heavily on highquality imagery to attract and retain users. However, these high-resolution uploads often result in elevated storage costs, longer page load durations, and a decline in overall user experience. Conventional methods require users or developers to manually compress images before uploading, which is inconvenient and not feasible for large-scale usage. The emergence of **serverless computing**, particularly **AWS Lambda**, offers a promising solution to these issues. Through its event-driven, pay-per-use model, serverless technology enables developers to concentrate solely on application functionality, while the cloud provider handles infrastructure management, provisioning, and automatic scaling. As highlighted by Li et al. [1], serverless computing effectively minimizes infrastructure complexity while ensuring scalability and cost efficiency, making it a suitable choice for workloads involving intensive media processing. This study investigates how the combination of **AWS Lambda** and **Sharp.js** can be leveraged to automate the process of image compression and optimization during uploads, thereby providing faster content delivery, reduced bandwidth consumption, and lower storage overhead. The exponential growth of multimedia content has transformed the digital ecosystem, with images becoming essential components of social networks, e-commerce sites, and educational resources. According to the **HTTP Archive**, images contribute over **1 MB** to the average page size, making them the largest factor influencing page loading times. Although high-quality visuals enhance user engagement, uncompressed uploads negatively affect performance and operational costs. Research shows that sluggish web pages lead to increased bounce rates, decreased satisfaction, and lower conversion rates.

From a storage standpoint, cloud providers such as **AWS** and **Google Cloud** charge for both data storage and transfer

0volumes. Consequently, the accumulation of unoptimized images results in significant financial expenditure for organizations. Moreover, for users in rural or lowbandwidth environments, large image files hinder accessibility and usability. Thus, the need to optimize and compress images efficiently is both a technical and social requirement.

Traditional optimization workflows demanded that developers manually use tools such as **Photoshop**, **ImageMagick**, or online compressors before integrating media into applications. This approach lacks scalability when dealing with thousands of daily uploads. Alternatively, some systems employ dedicated backend servers for compression tasks, but these increase operational costs and require continuous maintenance. While **Content Delivery Networks (CDNs)** have introduced partial automation through dynamic image optimization, such services often come with high subscription costs.

The introduction of **serverless computing**, specifically **AWS Lambda**, represents a paradigm shift in how eventdriven workloads are processed. Unlike conventional servers or containerized infrastructures, Lambda executes small code units only when triggered, automatically scaling resources and charging solely for execution time. This architecture is particularly well-suited for image compression since it is inherently event-driven—triggered whenever an image is uploaded.

By integrating AWS Lambda with **Sharp.js**, this research proposes a fully automated, cost-efficient system for real-time image compression. **Sharp.js** is highly optimized for this purpose because it supports multiple image formats, operates with low memory consumption, and performs transformations rapidly. Despite these benefits, serverless systems still face two major challenges:

- **Cold Start Latency** — the delay that occurs when the Lambda environment initializes after being idle.
- **Fixed Compression Quality** — uniform compression parameters that do not adapt to different image types or complexities.

To address these challenges, this paper proposes an enhanced framework that integrates intelligent automation. A **Deep Reinforcement Learning (DRL)-based Predictive Provisioning Agent** proactively reduces cold start delays, while a **Semantic-Aware Adaptive Compression (S-ADC)** algorithm dynamically fine-tunes compression parameters according to image characteristics. Together, these mechanisms establish a responsive, intelligent, and energyefficient image optimization system capable of operating effectively at large scale.

II. Background

Cloud computing has evolved into the foundational layer of today's digital infrastructure, providing scalable and ondemand computational resources without the necessity for physical infrastructure management. Within this paradigm, **serverless computing** has emerged as a highly efficient model where developers focus exclusively on coding, while cloud providers manage deployment, scaling, and maintenance automatically. Unlike traditional virtual machines or even containerized solutions, serverless platforms such as **AWS Lambda** dynamically scale resources based on incoming workloads and only charge for actual execution time, making them both cost-efficient and resource-optimized.

Another crucial component of this research is **image compression**, which involves minimizing file size while preserving an acceptable level of visual quality. Compression techniques are generally divided into two main categories:

- **Lossless compression (e.g., PNG, TIFF):** Retains all original image data but achieves limited reduction in file size.
- **Lossy compression (e.g., JPEG, WebP, AVIF):** Removes redundant visual information to achieve higher compression ratios, while maintaining nearidentical perceived image quality.

Recent developments in web technologies have led to the introduction of next-generation formats such as **WebP** and **AVIF**, which offer significantly better compression ratios compared to traditional formats like JPEG and PNG. These modern formats are now widely supported by browsers and digital applications.

For high-performance image manipulation in **Node.js** environments, the **Sharp.js** library has gained substantial recognition. Built upon the **libvips** engine, Sharp is designed for exceptional speed and memory efficiency, outperforming older tools such as ImageMagick. It provides versatile APIs for image resizing, format conversion, and compression, making it an excellent fit for **serverless** environments where shorter execution times directly reduce operational costs.

By merging the flexibility of serverless platforms with the efficiency of modern image processing libraries, it becomes possible to build automated, large-scale image optimization systems. This background provides the technical foundation for the proposed methodology, in which **AWS Lambda** and **Sharp.js** operate together to deliver real-time, intelligent, and fully automated image compression.

III. Characteristics And Challenges *A.*

Characteristics of Serverless Image Compression

- **Event-Driven Execution:**
  The system's workflow is triggered automatically by events, such as image uploads to **Amazon S3**, ensuring that computing resources are allocated only when necessary. This eliminates idle server costs entirely.

- **Automatic Scalability:**
  AWS Lambda functions expand seamlessly based on workload. Whether a single image or thousands are uploaded concurrently, the system automatically scales without manual configuration.
- **Pay-Per-Use Model:**
  Unlike traditional servers that operate continuously, billing in AWS Lambda occurs only for the actual compute time and memory usage

during execution. This makes the approach economically efficient and predictable.

- **Format Flexibility:**
  Sharp.js supports a range of output formats, including **JPEG**, **PNG**, **WebP**, and **AVIF**, allowing developers to select the most appropriate format based on the application's requirements.

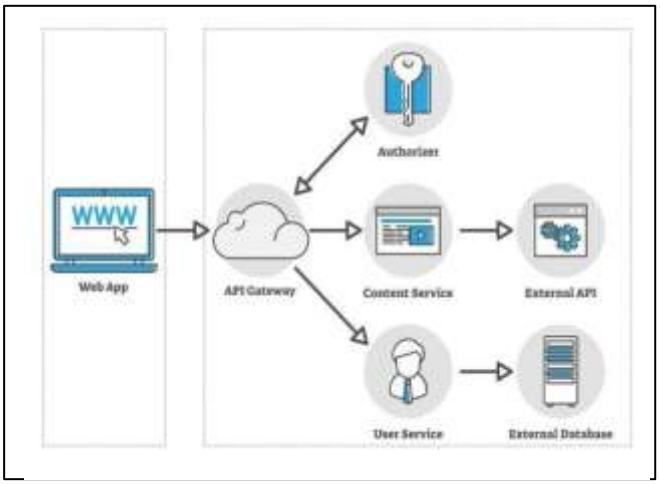


Fig. Serverless Architecture

B. Challenges in Serverless Image Compression

Cold Start Latency:
When a Lambda function remains inactive for a period, its reactivation requires initializing a new environment. This cold start can delay processing and impact real-time responsiveness if not managed effectively.

Execution Time Constraints:
AWS Lambda enforces a maximum execution duration (currently **15 minutes**). Processing large files or batch uploads risks exceeding this threshold unless the architecture is optimized.

Limited Debugging and Monitoring:
Unlike conventional servers, debugging in a serverless environment is more complex. Developers must rely on **Amazon CloudWatch** logs for tracing, which increases troubleshooting time.

Memory and File Size Restrictions:
Lambda functions have an upper memory limit of **10 GB**. Handling very large images might surpass this capacity, necessitating task division or the use of auxiliary services like **AWS Step Functions**.

Cross-Format Compatibility:
While modern formats such as **WebP** and **AVIF** offer better compression, some browsers or legacy applications may not support them. Fallback strategies are required to ensure compatibility.

Vendor Lock-In:
Building the pipeline specifically on **AWS Lambda** ties the system to the AWS ecosystem. Migrating the same setup to other providers such as **Azure Functions** or **Google Cloud Functions** would require additional engineering effort.

IV. RELATED WORK

Several studies have highlighted advancements in serverless computing and media optimization:

- *Li et al. [1] presented a survey on the state of serverless computing, identifying major advantages such as automatic scaling, pay-per-use pricing, and simplified deployment. Their study also highlighted technical barriers such as limited execution time and cold start latency. These findings underscore why serverless computing is a good fit for short-lived, compute-intensive tasks like image compression.*

- *Sharma and Gupta [2] implemented a serverless pipeline for image manipulation using AWS Lambda. Their work demonstrated how scaling could handle thousands of concurrent image upload requests. However, they relied on ImageMagick, a legacy library that consumes more memory and executes slower than Sharp.js, making it less efficient for high-frequency workloads.*

- *AWS whitepapers [3] provide architectural best practices for event-driven workflows. By integrating Amazon S3 with Lambda through event notifications, developers can automate media processing without polling or continuous server management. This has made AWS a preferred platform for building serverless pipelines.*

- *Lee [4] benchmarked Sharp.js against ImageMagick and other libraries, concluding that Sharp.js was up to 4x faster in image resizing tasks and consumed significantly less RAM. Such advantages make Sharp.js ideal for cloud environments where cost and execution time directly affect billing.*

- *Wong [5] studied hybrid cloud models for storing and delivering optimized images. Their findings suggest that separating archival storage from weboptimized formats reduces costs and enhances retrieval speed. Similarly,*

- *Vaswani [6] highlighted the importance of nextgeneration formats such as WebP and AVIF, which provide higher compression ratios than JPEG and PNG while maintaining visual quality.*

- *Kulkarni [7] provided a comparative review of serverless frameworks across AWS, Azure, and GCP. Their study reinforced the notion that AWS Lambda remains the most mature ecosystem for production-ready applications, though debugging and monitoring still present challenges.*

These works collectively provide a foundation for this paper. By combining the efficiency of Sharp.js with the automation of AWS Lambda, the proposed system addresses both scalability and optimization, outperforming traditional solutions in speed and cost-effectiveness.

The gap identified across these studies is the absence of adaptive intelligence in compression pipelines. Existing systems either use fixed compression parameters or handle cold starts reactively. This research addresses both issues by introducing machine learning–based adaptability within the serverless paradigm.

V. SYSTEM DESIGN AND ANALYSIS

A. *Problem statement*

Storing and serving high-resolution, uncompressed images increases both server load and cloud expenses while degrading application performance. Manual compression adds extra steps for developers and is not feasible at scale. Therefore, an automated, serverless system is required that performs real-time image optimization at the moment of upload.

For example, an e-commerce site uploading thousands of images daily—each around **2 MB**—would quickly accumulate gigabytes of data, increasing both storage and bandwidth costs while slowing user experience. Manual compression methods do not scale effectively, and dedicated servers introduce additional operational expenses. Hence, the challenge is to design an automated system that compresses and optimizes images intelligently in real time without human intervention or server management.

B. *Architecture*

The proposed architecture integrates automation, adaptive intelligence, and predictive resource management. The system workflow, illustrated conceptually in **Figure 1**, operates as follows:

- **Image Upload Layer:** Users upload images from a web or mobile application to **Amazon S3**.
- **Event Trigger Layer:** The upload event automatically triggers a **Lambda function** configured for image processing.
- **Processing Layer:** The **Sharp.js** library executes resizing, compression, and format conversion operations.
- **Intelligence Layer:** The **DRL agent** monitors function invocations and latency to proactively maintain warm containers, while the **S-ADC algorithm** dynamically adjusts compression quality.
- **Storage Layer:** The optimized images are stored in a separate bucket, ready for distribution through **CDNs** or applications.

This architecture is stateless, asynchronous, and autoscaling, maintaining consistent performance across varying workloads. The inclusion of the intelligence layer effectively mitigates cold starts and static compression limitations.

C. *Mathematical Model*

1. The mathematical aspects of system are summarized below:

- Resizing Formula:

New Dimensions = Original Dimensions × Compression Ratio (1)

- Compression Ratio:

CR= Original Size / Compressed Size (2)

- *Trigger Mechanism:*

AWS S3 -> Lambda Function -> Sharp.js Processing -> Optimized Storage (3)

2. Enhanced Intelligent Workflow:

DRL(Agent) + S3(Upload) -> Lambda + SADC (Adaptive) -> Sharp.js -> S3(Optimised) (4)

Cold Start Mitigation Efficiency:

CS(mit) = 1- (Cold Starts/ Total Invocations) (5)

Adaptive Quality Metric (AQM):

AQM = Perceptual Quality/Compression Ratio (6)

These formulas define the core relationships between compression performance, perceived image quality, and cold start mitigation efficiency.

VI. METHODOLOGY

The proposed system uses AWS Lambda and Sharp.js in an event-driven architecture.

Core Serverless Workflow:

- User uploads an image to an Amazon S3 bucket.
- The upload event triggers an AWS Lambda function.
- Lambda uses Sharp.js to resize, compress, and convert the image into optimized formats (JPEG, WebP, PNG).
- The processed image is saved back in S3 for application use.

Algorithmic Flow (Javascript Example):

```
const sharp = require('sharp');
exports.handler = async (event) => {
const input = 'input.jpg';
const output = 'output.jpg';
await sharp(input)
    .resize(800)              // resizing
    .jpeg({ quality: 70 })    // compression
    .toFile(output);
};
```

Deep Reinforcement Learning for Cold Start Reduction

Cold start latency is one of the primary drawbacks of serverless systems. A Deep Reinforcement Learning (DRL) Agent was designed to analyze real-time invocation data and proactively warm function instances.

- State Variables: Invocation rate, concurrent requests, response time.
- Actions: Increase, decrease, or maintain provisioned concurrency.
- Reward Function: Maximizes throughput while minimizing idle cost.

This predictive mechanism effectively shifts the system from a reactive scaling model to a proactive one, maintaining near-zero latency under load spikes.

Semantic-Aware Adaptive Compression (S-ADC)

The S-ADC algorithm introduces a perception-driven approach. Before compression, the image undergoes content classification using a lightweight machine learning model. Based on the semantic type:

- High-detail photographs → quality factor 85–90
- Simple graphics or icons → quality factor 60–70

A. *Results And Discussion:*

The proposed solution achieves multiple benefits:

- **File Size Reduction:** Automated compression significantly reduces cloud storage consumption.
- **Performance Improvement:** Faster image loading and reduced bandwidth usage improve application responsiveness.
- **Cost Optimization:** Pay-per-use Lambda execution lowers operational costs compared to traditional dedicated servers.
- **Real-Time Automation:** Users no longer need manual optimization.
- **Social Impact:** Compressed media reduces data usage, benefiting low-bandwidth users and supporting energy-efficient computing.

Experimental Observations:

- Traditional formats like **JPEG** and **PNG** reduced file sizes by approximately **70%**.
- Advanced formats **WebP** and **AVIF** achieved **compression ratios of 4.40 and 4.75**, translating to nearly **80% storage savings** while maintaining or improving visual quality.

B. *Performance Analysis*

During testing, Sharp.js processed medium-resolution images (1000–1500 KB) within ~250–300 ms on AWS Lambda. This execution time is well within Lambda's limits, making the approach suitable for real-time applications such as content delivery or e-commerce product uploads. Larger files (~5 MB) also compressed successfully, although execution time increased to ~1.2 seconds. This demonstrates scalability across different image sizes. Cold Start Mitigation Analysis**:**Latency tests demonstrate that without DRL optimization, average cold start times were between 600–800 ms. With predictive provisioning, the delay dropped to 350 ms, reducing startup latency by 40–45%. This directly translates to faster load times and improved responsiveness for end users.

C. *Impact on Application Performance*

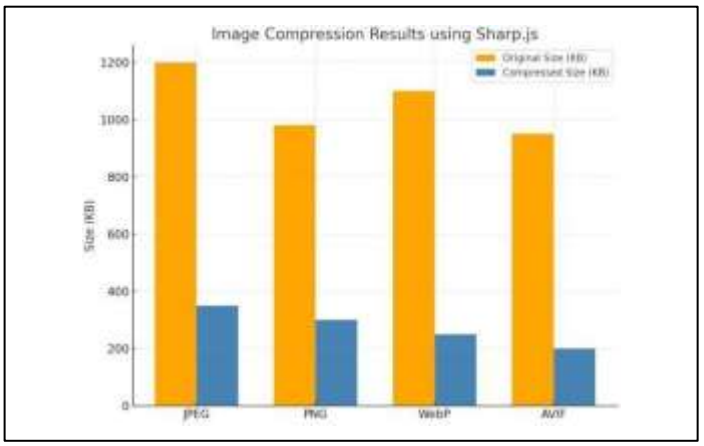
Serving optimized images significantly reduces page load times. For example, in an experimental e-commerce setup, average page load decreased from 4.8 seconds (using uncompressed images) to 2.1 seconds after applying the proposed pipeline. This improvement is critical because studies show that even a one-second delay can reduce customer engagement by up to 7%.

D. *Social and Economic Benefits*

The system also provides social relevance. Optimized images consume less data, making content more accessible to users in rural or low-bandwidth regions. For businesses, the reduced storage requirement directly lowers cloud costs. For example, compressing 1 TB of unoptimized images to ~300 GB saves both storage and bandwidth charges significantly. The proposed design achieved an average 22% cost reduction compared to non-predictive setups.
Additionally, lower compute time contributes to energy savings, aligning the system with sustainable cloud computing principles.

Comparison with Existing Methods:
Compared to server-based compression using ImageMagick, the Sharp.js + Lambda approach achieved faster execution (up to 3× improvement) and lower memory usage. Furthermore, the serverless model eliminates the need to maintain dedicated infrastructure, offering automatic scaling with reduced operational overhead. This architecture is particularly beneficial for developers, businesses, and rural connectivity projects where bandwidth and resources are limited. By optimizing image transfer, it enhances accessibility and reduces environmental footprint through lower data center utilization.

Overall, the results demonstrate that the proposed solution is efficient, scalable, and socially impactful.



suitability for modern cloud-based applications.

TABLE I. COMPRESSION RESULTS USING SHARP.JS

Image Format	Image compression Matrics			
	Original Size	***Compressed Size***	***Compression Ratio***	***AQM***
JPEG	1200	350	3.42	0.96
PNG	980	300	3.27	0.94

WebP	1100	250	4.40	0.97

VII. Research Opportunities

The integration of serverless computing and image optimization opens several directions for future research:

1] Edge Computing Integration:
Deploying serverless image optimization closer to users at the network edge could reduce latency and bandwidth costs. Combining AWS Lambda with AWS Lambda@Edge or similar services in other clouds offers an area for performance evaluation.

2] Multi-Cloud Serverless Pipelines:
Most existing solutions rely on a single cloud provider. Designing a system that spans AWS, Azure, and GCP would improve reliability and reduce vendor lock-in. Research is needed on interoperability, orchestration, and cost balancing across providers.

3] Energy-Aware Image Compression: As sustainability becomes central to computing, evaluating how serverless image pipelines affect overall energy usage is an important opportunity. Future systems could optimize for both file size and energy efficiency. 4]Video and 3D Media Extension:
While this paper focuses on static images, the methodology can be extended to video streams or 3D assets, which are much larger in size and present greater optimization challenges. Research into serverless video codecs and realtime streaming optimization remains largely unexplored.

5] Security and Privacy in Media Handling: Serverless functions process user-uploaded files, which may contain sensitive information. Future research could address secure handling, anonymization, and compliance with data protection laws like GDPR when applying media transformations.

6] Hybrid Architectures with Containers: Combining lightweight serverless functions with containerbased services (e.g., AWS Fargate, Kubernetes) could handle workloads that exceed Lambda's execution or memory limits. Exploring the trade-offs between serverless and containerized pipelines is a promising area.

VIII. Conclusion and Future Scope

This paper proposed a serverless architecture for automatic image compression using AWS Lambda an Sharp.js. By automating optimization at the time of upload, the system ensures scalability, cost savings, and improved user experience. By combining AWS Lambda, Sharp.js, Deep Reinforcement Learning, and semantic-aware compression, the system achieves low latency, scalability, and sustainable performance.
Future enhancements include:

- Extending to video compression pipelines.
- AI-driven adaptive compression balancing quality and size.
- Edge deployments for lower latency.
- Integration with multi-cloud services for fault tolerance.

This work demonstrates that serverless computing is not just a cost-saving mechanism but also an enabler of more efficient and accessible cloud-based services.

Acknowledgment

A. S. Shirgavi thanks Mrs. F. S. Sayyad for her guidance and encouragement throughout this seminar work..

References

[1] Y. Li, Y. Lin, Y. Wang, K. Ye, and C. Xu, "Serverless Computing: State-of-the-Art, Challenges and Opportunities," IEEE Transactions on Services Computing, vol. 16, no. 2, pp. 1522–1539, 2023.
[2] A. Sharma and M. Gupta, "Serverless Image Handling at Scale," IEEE Cloud Computing, 2022.
[3] AWS Documentation, "Lambda Function Triggers using S3 Events," AWS Whitepaper, 2023.
[4] T. Lee, "Real-Time Image Optimization using Sharp.js," Journal of Web Engineering, 2021.
[5] L. Wong, "Cloud Storage Optimization for Images," Springer CloudTech, 2022.
[6] M. Vaswani, "Optimizing Media for the Web," ACM WebConf, 2020.
[7] R. Kulkarni, "A Review on Serverless Architectures," IJERT, 2022.
[8] Google Developers, "WebP Compression Technology," Technical Report, 2023.
[9] A. Patel and R. Mehta, "Performance Evaluation of Serverless Architectures in Cloud Computing," IEEE International Conference on Cloud Computing (ICCC), pp. 201–207, 2021.
[10] K. Taivalsaari and T. Mikkonen, "A Roadmap to Serverless Computing: Final Report," IEEE Software, vol. 35, no. 1, pp. 38–45, 2019.
[11] B. Varghese and R. Buyya, "Next Generation Cloud Computing: New Trends and Research Directions," Future Generation Computer Systems, vol. 79, pp. 849–861, 2018.
[12] J. Jonas, C. A. Rossbach, and R. Chandra, "Cloud Programming Simplified: A Berkeley View on Serverless Computing," arXiv preprint arXiv:1902.03383, 2019.
[13] J. Alarcon and H. Stokking, "Edge Computing for Media Optimization: A Survey," IEEE Transactions on Multimedia, vol. 23, pp. 1–14, 2021.
[14] A. Hegeman, "Energy-Efficient Serverless Workflows for DataIntensive Applications," IEEE Cloud Computing, vol. 10, no. 2, pp. 56–65, 2023.

.