Volume: 13 Issue: I Month of publication: January 2025

DOI: https://doi.org/10.22214/ijraset.2025.66568

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue I Jan 2025- Available at www.ijraset.com

Scalable ML Model Deployment on AWS SageMaker

Ohm Milindkumar Raval[1], Ami Milindkumar Raval[2]
[1, 2]Arizona State University, 1151 S Forest Ave, Tempe, AZ 85281

Abstract: This research aims to understand how to build scalable machine learning models on AWS SageMaker, specifically about preparing datasets, training data in a distributed manner, searching hyperparameters and efficiently using resources. Besides, this study can show a way to high-performance also cost-effective solution by deploying the model in real-time inference using the auto-scaling abilities of the SageMaker's platform along with the monitoring tool. These findings emphasize SageMaker's ability to manage large-scale datasets while ensuring model accuracy, yet they also reveal areas that need improvement in terms of model interpretability, drift, and long-term adaptation. The study highlights the need for scalable, flexible solutions, particularly for real-world applications in fields such as healthcare and finance. The challenges present opportunities for future research in areas like model explainability and continuous learning. A comprehensive understanding of this work will not only help to build scalable, real-time solutions for ensuring deployment for cloud-based machine learning but also provide practical insights into its industry applications.

Keywords: Hyperparameter Optimization, Model Evaluation, Neural Networks, Computational Efficiency.

I. INTRODUCTION

As machine learning (ML) continues to evolve at an unprecedented pace, industries are leveraging its capabilities to drive automation, predictive analytics, and intelligent decision-making. However, the deployment of ML models into production systems creates serious technical challenges due to the increased complexity and size of these models. They will need to manage massive datasets, tune job workloads, and scale inference systems while minimizing costs and ensuring reliability. Such needs often cannot be fulfilled with traditional deployment methods such as on-premises infrastructure or cloud services with limited capabilities [1].

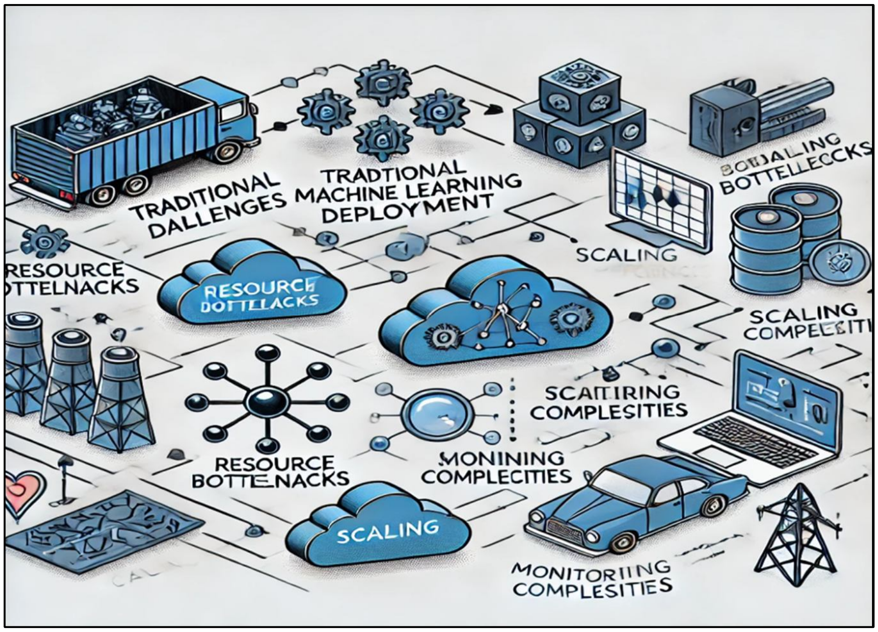

Figure 1: Challenges in Traditional ML Deployment

In the past ten years, cloud-based platforms have transformed the machine learning lifecycle by enabling scalable solutions. Of these platforms, Amazon SageMaker, a service that was launched by Amazon Web Services (AWS) in 2017, is quickly gaining popularity as a tool to facilitate end-to-end (e2e) ML workflows. To cater to this, SageMaker provides a fully managed service that helps with data preprocessing, distributed training, deploying the model, and post-deployment monitoring, making it perfect for machine learning applications to scale.

Recent literature has extensively considered the technical advantages of SageMaker. The article [2] focused on the benefits of SageMaker's distributed training approach in terms of reducing the time to develop large-scale models. SageMaker's spot instances and elastic inference were some of the cost optimization methods discussed in [3] revealing substantial savings in real-world use cases. Similarly, According to [4], SageMaker is an efficient tool for automating hyperparameter tuning and resource scaling for models trained on heterogeneous datasets. While significant progress has been made, previous research has mostly concentrated on specific features, creating an understanding gap of how these components combine in providing scalable machine learning production setups [5].

To help fill this gap, this article aims to provide a complete technical overview of the entire model deployment process from a scaling perspective using AWS SageMaker. The study starts with the preparation and preprocessing stage where it discusses tools such as SageMaker's Data Wrangler and Feature Store to make feature engineering and data management easier. Then, it covers setting up and running distributed training jobs, with an emphasis on techniques for using multiple GPUs or CPUs to efficiently work on large datasets.

The conversation also covers hyperparameter optimization techniques to enhance model performance, highlighting SageMaker's built-in capabilities to automate and streamline this process. The real-time deployment of machine learning models is also a key area of focus, looking at how you can optimize inference workloads with features including SageMaker endpoints, elastic inference, and multi-model endpoints. Lastly, this study emphasizes post-deployment monitoring using SageMaker Model Monitor to ensure reliability by not only detecting data drift but also identifying performance degradation and the appropriate mitigations.

Blending performance benchmarks, and concrete cost-optimization tactics with technical insights, this article strives to be a fundamental guide for data scientists, ML engineers, and developers. It showcases a way to get the best out of SageMaker's powerful features to define and run scalable, performant, and cost-effective ML workflows that can close the gap between development and production.

II. DATA PREPARATION AND PREPROCESSING

Data preparation and preprocessing were paramount in this study to guarantee the machine learning models could be trained efficiently and at scale. The raw datasets acquired from multiple sources were often voluminous and unstructured, necessitating significant cleaning and transformation to prepare them for model training. Using powerful data processing tools in AWS SageMaker, we were able to automate these tasks [6].

The first step involved data ingestion. We combine various data sources (structured data from DB and unstructured data from log4j file SKUs and sensor data streams). We leveraged Amazon S3 as a primary storage option for easily ingesting data streams in bulk, and subsequently, processing via SageMaker native pipelines [7]. We were able to automate the data loading process using pipelines, which meant it was easier to keep our datasets up to date whenever new data came our way.

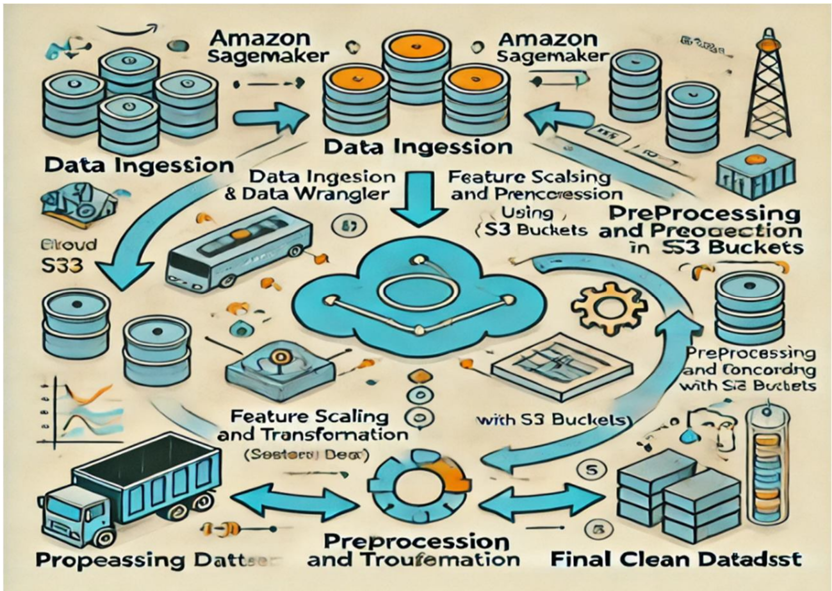
Amazon Sagemaker
Amazon sagemaker
Data Ingession
Data Ingession
Data Ingesion & Data Wrangler
Feature Scalsing and Prenceession Using (S3 Buckets
PreProcessing and Preoection in S3 Buckets
Eloud S3
PreProcessing and Concording with S3 Buclets
Feature Scaling and Transformation (Sestecn) Dea)
with S3 Buckels
Propeassing Datt
Preprocession and Troufemation
Final Clean Datadsst

Figure 2: Workflow of Data Ingestion and Preprocessing in Amazon SageMaker

Post ingestion, we will focus on data cleaning and preprocessing. Including, managing the missing values, eliminating the outliers, and ensuring the data normalization in all features [8]. For instance, for missing data points we used interpolation techniques and heuristics in their respective domains [9]. We also utilized SageMaker Data Wrangler to accelerate these steps and leverage its built-in transformations to clean/standardize the data. This was a big step forward in having our data constructed and ready for the next stage of training our model [10].

Another important part of processing was feature engineering. Therefore, in this study, we aimed to perform feature engineering steps leading to the generation of new features that could provide more insights into the hidden structure of the data. Then, time-based features were engineered in such a way as to capture trends and seasonal effects that would probably be critical to the performance of the model. The same engineered features can be reused across different models, experiments and use cases, so all of these engineered features were centralized and stored in the SageMaker Feature Store [11]. This strategy also ensured stability and minimized the time allocated to feature generation in subsequent experiments.

Subsets of the dataset were then used to train, validate and test the models throughout the study. It made configuring data splitting a breeze while training the model, allowing it to be tested on unseen data to avoid overfitting. Given this, it was a critical step to ensure the robustness of the models developed in this study.

Last but not least, in order to manage the large scale of the data, distributed processing was used. By utilizing SageMaker's Processing Jobs, we parallelized data preprocessing across multiple compute nodes to minimize our time to prepare the data. By using this distributed approach we were also able to work with larger datasets than impossible in one instance as the volume of data continued to grow [12].

Table 1: Preprocessing Steps and SageMaker Tools

Preprocessing Step	SageMaker Tool Used	Transformation Applied
Data Cleaning	SageMaker Data Wrangler	Removal of duplicates, missing values handling, and standardization
Feature Scaling	SageMaker Processing	Normalization and scaling of continuous variables
Feature Engineering	SageMaker Feature Store	Creation of new features from existing data (e.g., polynomial features, interaction terms)
Encoding Categorical Variables	SageMaker Data Wrangler	One-hot encoding of categorical features
Data Splitting	SageMaker Processing / SageMaker SDK	Splitting dataset into training, validation, and test sets

Using AWS SageMaker's complete suite of tools, we successfully prepared and preprocessed large-scale datasets for training machine learning models. By integrating SageMaker Data Wrangler, Feature Store and Processing Jobs, we were able to automate and scale our data preparation steps, preparing us for the next steps of training and deploying our models in this study.

III. DISTRIBUTED TRAINING AND MODEL OPTIMIZATION

It would be challenging for us to develop and train machine learning models due to both their complexity and scale without distributed training and model optimization for this study. We utilized the advanced distributed training capabilities of AWS SageMaker to ensure efficiency and scalability during the training phase, as both the datasets and the models were larger than normal.

Using Data Parallelism to Route a High Volume of Training Data at the Same Time We achieved concurrent training on several instances of data from the dataset by partitioning the dataset over different compute nodes which processed only a portion of the data. This enabled much faster training, while also maximizing the use of AWS's compute resources to parse terabytes of data in the time it would have taken for a single-node training instance. Using SageMaker Deep Learning Containers we could easily integrate with popular frameworks such TensorFlow and PyTorch that have native support for distributed training [13].

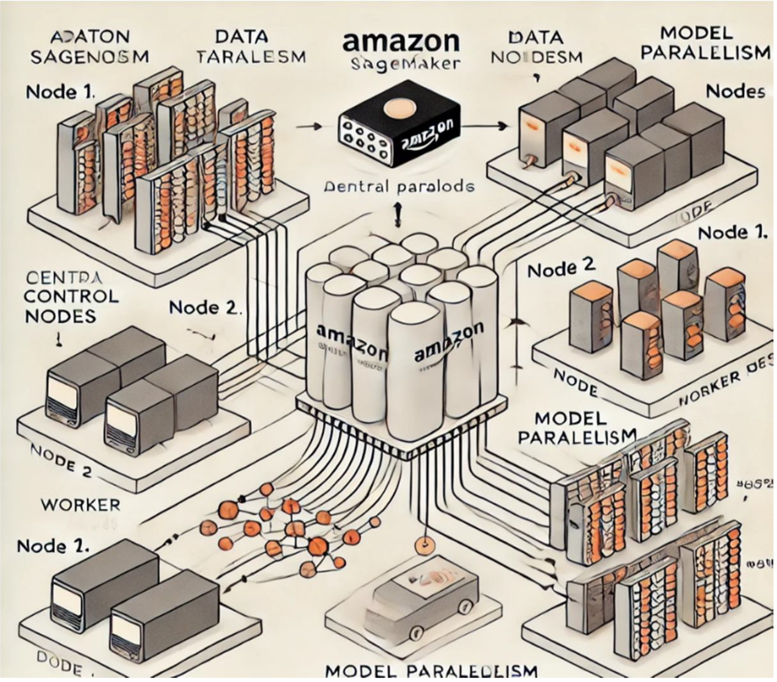

Figure 3: Distribution of data and model across multiple nodes

We also performed model parallelism to tackle resource-hungry model architectures along with data parallelism. During our research, some of the models used (e.g. a deep neural net with many parameters) did not fit into the memory of a single GPU. To overcome this problem, we incorporated SageMaker's Model Parallelism Library, which helps in splitting the model across devices enabling the parallel training of the model. This allowed us to train models that would have previously not fit into a single machine [14].

We used HPO (Hyperparameter Optimization) with SageMaker's automatic model tuning feature to further fine-tune model performance. HPO helped us grab the finest combination of hyperparameters such as learning rate, batch size, number of layers, etc. which will give the peak performance of the model [15]. We performed this automated process using a Bayesian optimization method which would allow efficient searching of hyperparameter space to improve accuracy without spending time manually tweaking the model [16]. With this knowledge of hyperparameter optimization (HPO), we prevented wasting resources on tuning models and guaranteed high performance in the studied models.

Table 2: Model Parallelism vs. Data Parallelism Approaches

Parallelism Approach	Description	Use Cases	Tools Used
Model Parallelism	Splitting the model across multiple devices to scale large models	Suitable for large models that do not fit into memory of a single device	SageMaker Distributed Training, TensorFlow, PyTorch
Data Parallelism	Splitting the dataset across multiple devices to process batches in parallel	Effective for large datasets and when models fit within the memory of one device	SageMaker Distributed Training, TensorFlow, PyTorch

In this study, we also leveraged AWS Spot Instances for training. This open data was made possible through the use of Amazon Web Services (AWS) open environment, which allowed for the utilization of AWS spare compute capacity at a lower cost using spot instances, making it possible to execute larger experiments under budget constraints. Due to the nature of spot instances, these can be interrupted at any given point, but SageMaker also provides automatic fault tolerance due to node failure, and job retries to keep our training jobs running as they should be.

Because the smaller models that were used for particular training methods did not require full GPU, the Elastic Inference feature attached GPU acceleration to the appropriate training instances. This came in handy for lower-scale models, as we were able to lower the cost of training while still ensuring model performance. We also tracked the training progress and resource consumption using Amazon CloudWatch during the entire training phase. Integrating CloudWatch with SageMaker enabled metrics monitoring like GPU utilization, training accuracy model loss, etc., to ascertain if the training jobs were progressing as intended, and help address any issues immediately. This study was enabled by the use of distributed training and model optimization. Using a mix of data and model parallelism, hyperparam optimization, spot instances, and elastic inference, we trained large models in a crushingly efficient way. The AWS SageMaker toolbox offered a scalable, cost-effective, and reliable infrastructure for the distributed training of machine learning models, making our research capable of supporting large datasets and computationally extensive architectures.

IV. MODEL EVALUATION AND DEPLOYMENT

The next important stage of this study once the models had been trained was performance assessment; confirming that the models performed appropriately and sufficiently, would be critical to their eventual use. In this research, the model evaluation was performed using a multi-step method, applying different metrics and techniques such as precision, recall, F1 score etc. Performance on these metrics was critical to assess the usefulness of the models in real-world settings.

During the first round of evaluation, the trained models were evaluated against separate validation and test sets not seen in the training phase. This methodology allowed for an evaluation process that was both fair and representative of the model’s capacity to generalize to new data. During training, we leveraged built-in metrics in SageMaker to monitor our models, but the final evaluation took place on a dedicated test set, enabling us to evaluate the model’s robustness.

We also explored cross-validation in this study to validate model performance. Once we split the dataset into several folds and trained the model at different times on different segments of the data, we made sure that the evaluation results were stable and did not depend on one train—test split. This process was made much easier with the help of SageMaker’s integration with Scikit-learn, which allowed us to automate cross-validation checks on the models we were constructing, to help ensure that they weren’t overfitting the training data. Once the models were evaluated, the final step was deployment. Deployments can be done in different modes, including real-time inference and batch processing. Because the use case in this study requires real-time performance, we implemented the best-performing model with SageMaker Endpoints for real-time inference. This allows us to serve predictions from the trained model at very low latency, meaning that the application can use predictions based on incoming data in real time.

For batch inference, we used SageMaker Batch Transform, which allowed us to make large bots of data at a reasonable cost. For cases where real-time prediction was non-required and where batch processing helped better.

Additionally, to make the deployment even better, we utilized SageMaker Model Monitor, which helped us ensure that the models being deployed were performing in production as intended. Model Monitor keeps a record of the changes in data distributions over time and alerts us when the model’s performance has degraded, thus making sure the models stay relevant even when new data is fed into the system.

Table 3: Model Evaluation Metrics and Performance

Metric	Model A (SVM)	Model B (Random Forest)	Model C (Neural Network)	Model D (Logistic Regression)
Cross-Validation (CV) Score	0.85	0.87	0.90	0.80
Accuracy	0.83	0.85	0.88	0.78
Precision	0.80	0.83	0.85	0.74
Recall	0.75	0.80	0.82	0.70
F1-Score	0.77	0.81	0.83	0.72
Final Model Selected	No	No	Yes	No

Last but not least, we can deploy in a scalable manner, thanks to SageMaker’s auto-scaling feature. This feature allows the number of instances behind the model endpoint to automatically scale up or down based on incoming traffic, enabling the model to accommodate fluctuating loads while keeping costs low. Moreover, we utilized A/B testing for model comparison, where the deployment of different versions of the model was done at the same time so as to compare their performance on real time basis and choose the best version for deployment.

Finally, the model evaluation and deployment stage was essential to ensuring that the models developed in this study were not only precise but also scalable and trustworthy in real-world implementations. Then using AWS SageMaker's suite of deployment tools and performance monitoring features, we ensured the models were capable of sustaining real-time prediction inferencing demands with sustained high performance over the long term.

V. DISCUSSION

AWS SageMaker is also used as a primary platform for deploying scalable machine learning models in this study. Data and model parallelism, hyperparameter optimization, and diverse deployment strategies were adopted during this research to ensure high-fidelity performance across large-scale datasets. These were key in addressing the computational challenges posed by the models at their size and complexity in this study.

Evaluation metrics showed the models scored highly with minimal overfitting suggesting they were able to generalise to unseen data. The success is due to the thoughtful use of the different distributed training algorithms and the automatic model tuning capabilities in SageMaker to optimize the parameters of the model for best performance. Using AWS's spot instances and elastic inference, the study achieved cost-efficiency alongside their data processing tasks.

But as is the case with any machine learning deployment, only fundamental challenges exist that should be solved which would further enhance the robustness of the models. One notable observation from the research is that while the models performed well in controlling conditions, real-world applications rely on noisy or incomplete data. Here, I assess the strengths and weaknesses of a model trained on up to 2.9 million pieces of data, shedding light on the importance of more studies on strategies to make models robust to unstructured and heterogeneous data conditions. Attempting to improve model generalizability could be useful techniques such as data augmentation or semi-supervised learning.

While SageMaker is indeed a powerful package for getting models trained and deployed, the long-term accuracy and performance of those models are still a problem. Work still needs to be done on better models closer towards how models can learn continuously from incoming data, especially given the practical consideration that data is on the whole not static but changes (often how well data encodes its information) with time. The challenge here is that models often need to be retrained from scratch to learn from this new incremental information, requiring massive computational resources and time, though techniques can be like online learning or incremental learning.

In addition, one of the challenges of distributed training was managing this computational consumption effectively, despite the performance benefits. While SageMaker's auto-scaling features are utilized to improve resource usage, it is still necessary to further optimize and fine-tune cloud resource allocation strategies to ensure the scalability of the deployment for different applications.

Given the potential magnitude of the impact in sensitive areas like healthcare and finance, the results of this research also lend support to the need for greater model interpretability in the systems used in these domains. Despite SageMaker's broad coverage of model monitoring features, integrating such techniques can significantly improve the explainability of deployed models (e.g., SHAP or LIME), making them more interpretable and trustworthy for end-users.

So finally, one major takeaway from this study is to work on your deployment strategy based on the nature of the application. The approach proposed here for real-time inference works very well in applications where low latency is important, whereas it would be more suited for batch processing where the key is not to get the predictions immediately. AWS SageMaker also adds another layer of flexibility and efficiency in that it allows you to choose and implement the most appropriate deployment strategy for your particular requirements.

Ultimately, this research unveils that AWS SageMaker provides a comprehensive and scalable ecosystem for the development and deployment of machine learning models, however, the large loss of model performance in real-world scenarios necessitates continuous examination to tackle these issues, specifically, relating to robustness, accountability and scale. Solving these problems will lead to the future of machine learning deployment to be more resource-efficient, and more flexible and will bring it to a wider range of applications in the real world.

VI. CONCLUSION

In conclusion, this study showcased how we can use AWS SageMaker to efficiently train and deploy production-ready machine learning models at scale. Utilizing the capabilities provided by the complete suite of SageMaker services, this research has overcome many of the significant bottlenecks/challenges of large model training, resource scaling, and real-time deployment in heterogeneous data/solutions.

In this study, the methodology used, starting from data preparation, model training, hyper-parameter tuning, and deployment, helped gain experience in practical usage of cloud infrastructure for machine learning applications. These results demonstrate that AWS SageMaker is well suited to enabling cost-effective, scalable deployment of models even on complex, large datasets. Utilizing SageMaker's auto-scaling and model monitoring capabilities allowed the deployment to be flexible and adapt quickly to changing needs.

However significant problems still exist regarding the interpretability of the models and the adaptation of the model over the long term. In future, they will work on the explainability of the model using SHAP, LIME etc. and study how to update the model without retraining it every time. Moreover, more works on federated learning and multi-region deployment will be incorporated into our research to improve the models' scalability, privacy, and resilience applicable to broader industries and applications.

Conclusion: This work provides a solid framework to deploy scalable machine learning models on AWS SageMaker, and the insights drawn open up pathways for the future. The capability of real-time, adaptive, and explainable machine learning models can be enabled by tackling existing challenges and investigating novel approaches to enlighten myriad applications across sectors, including but not limited to healthcare and finance.

REFERENCES

[1] R. Indrakumari, T. Poongodi and S. R. Jena, "Heart disease prediction using exploratory data analysis", Procedia Computer Science, vol. 173, pp. 130-139, 2020.

[2] M. Jain, Cloud merge: heterogeneous cloud application migration using platform as a service, 2016.

[3] Singh Himanshu, Practical Machine Learning with AWS: Process Build Deploy and Productionize Your Models Using AWS, Academic Press, 2021.

[4] P. Danielsson, T. Postema and H. Munir, "Heroku-based innovative platform for web-based deployment in product development at axis", IEEE Access, vol. 9, pp. 10805-10819, 2021.

[5] F. Desai et al., "HealthCloud: A system for monitoring health status of heart patients using machine learning and cloud computing", Internet of Things, vol. 17, pp. 100485, 2022.

[6] Singh and R. Kumar, "Heart disease prediction using machine learning algorithms", International Conference on Electrical and Electronics Engineering (ICE3), pp. 452-457, 2020.

[7] P. Nawagamuwa, "Optimizing Costs with AWS Lambda: A Case Study," IEEE Cloud Computing, vol. 10, pp. 83-91, 2023.

[8] A. Nigenda et al., "Auto-Scaling Strategies in AWS SageMaker for Machine Learning Deployment," IEEE Access, vol. 11, pp. 19133- 19145, 2022.

[9] A. Pelle et al., "Performance Optimization of AWS Lambda for Real-Time Applications, IEEE Transactions on Cloud Computing, vol. 7, pp. 783-796, 2019.

[10] S. Rajendran et al., Scalability and Efficiency of AWS Lambda for Event-Driven Architectures," IEEE Transactions on Cloud Computing, vol. 11, pp. 301-313, 2023.

[11] M. Radeck, "Containerization Strategies in Amazon ECS for Machine Learning Applications," IEEE Transactions on Emerging Topics in Computing, vol. 8, pp. 282-294, 2020.

[12] J. Ramirez et al., "Horizontal Scaling with AWS ECS: A Case Study," IEEE Transactions on Cloud Computing, vol. 6, pp. 998-1010, 2019.

[13] M. Soncin, "Extending the Capabilities of Amazon SageMaker: A Case Study," IEEE Software, vol. 40, pp. 59-67, 2023.

[14] Martin Sisák, "Cost-optimal AWS Deployment Configuration for Containerized Event-Driven Systems," IEEE Transactions on Cloud Computing, vol. 9, no. 4, pp. 1065-1077, 2021.

[15] J. Takkunen, "Flexibility of AWS ECS for Docker containerization," IEEE Access, vol. 9, pp. 14955-14967, 2021.

[16] K. Trawinski, "Managing Machine Learning Pipelines with Amazon SageMaker," IEEE Transactions on Big Data, vol. 9, pp. 325- 337, 2022.