International Journal of Computer Trends and Technology
ISSN: 2231–2803 / https://doi.org/10.14445/22312803/IJCTT-V72I12P119

Volume 72 Issue 12, 153-163, December 2024


*Original Article*

# Serverless AI-Powered Recommendation Engine with AWS Lambda and SageMaker

Joyanta Banerjee[1], Soumya Barman[2]

[1]*Senior Modernization Architect, Amazon Web Services, Santa Monica, CA, USA.*
[2]*Senior Cloud Infrastructure Engineer, Mckinsey and Company, Seattle, WA, USA.*

[1]*Corresponding Author : joyanta.banerjee@gmail.com*

Received: 05 November 2024 Revised: 30 November 2024 Accepted: 18 December 2024 Published: 31 December 2024

***Abstract*** *- The opportunities for e-commerce, entertainment, and many other services, which can be provided through the internet, have stimulated the growing need for recommendation systems to improve the experience of users. Such recommendation systems are based on complex matrices and need substantial equipment with high maintenance costs; therefore, they are hindered by scalability and performance constraints. Aims: We want to show how organizations could use serverless computing and leverage the exponential development of artificial intelligence to manage the scalability of effective recommendation systems with ease of deployment and usually low operating costs. Study Design: This paper describes the architecture and development of a serverless recommendation system for an e-commerce application based on AWS Lambda and SageMaker. The potential of the serverless to reduce costs, scale automatically, and be deployed and maintained easily is also investigated. Furthermore, we incorporate Amazon SageMaker for training, deploying, and managing machine learning models behind the recommendation engine. Place and Duration of Study: Organizations across various industries have implemented this approach in 2023 and 2024. Methodology: Collaborative, content-based filtering and the hybrid approach are employed in the recommendation process, and the results are generated in real-time. The complete application is built using the serverless computing model, in which AWS Lambda runs simple code in response to events or user interactions. In contrast, Amazon Sage Maker is used to train the models and make predictions. Exposing APIs is done with AWS API Gateway; storing users' data is done with Amazon DynamoDB, while the model artifacts and the big data are stored in Amazon S3. Results: This architecture helps to avoid provisioning and managing servers, which makes the operation less complex. In this paper, we will describe all stages of the work, from data preprocessing to the generation of recommendations. Conclusion: The results thus demonstrate the exceptional scalability and responsiveness of the recommendation engine, capable of accommodating users' real-time needs with trivial time delay.*

***Keywords*** *- Serverless computing, AWS lambda, Amazon sagemaker, Machine Learning, Collaborative filtering, Content-based filtering, Scalability.*

## 1. Introduction

Recommendation systems are now incorporated into nearly every contemporary online service, with special emphasis on e-commerce, media hosting, and social networking. The way that they have been able to increase engagement and customer satisfaction, and thus increase revenues, has been through personalizing the messages to the users. Central to these systems are machine learning personas, which analyze user behavior to recommend products or content, among other things. Conventional recommenders are designed in a centralized setup where infrastructure has to be scaled up by hand and is expensive to maintain. [1-4] However, with the latest trends in serverless computing, companies like Amazon provide serverless computing solutions like AWS Lambda and Amazon Sagemaker for developing recommendation engines in a better way. Serverless computing is a model in which applications are executed without the provision of servers, implementing overhead minimization with provision for scaling as per requirements. In this research paper, an extensive framework for creating an AI-based recommendation system utilizing AWS Lambda and Amazon SageMaker is developed.

The goal of this architecture is a low-cost, scalable, real-time recommendation engine that does not require constant updating of the underlying application infrastructure to reflect user behavior. This is made possible by embedding Services such as serverless technologies coupled with the use of Machine learning algorithms to make Smart personalized recommendations.

### *1.1. Evolution of Recommendation Techniques*

Thus, the development of recommendation approaches has been through considerable progress in the years due to technological advancement, the shift in customer behavior and the complex data set. Achieving this understanding is crucial for analyzing the current status of and future prospects for recommendation systems.

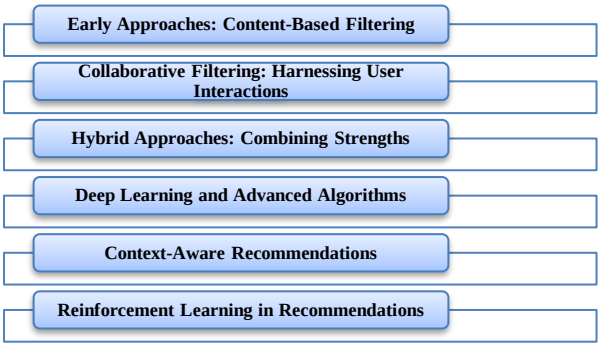


**Fig. 1 Evolution of recommendation techniques**

*1.1.1. Early Approaches*

Content-Based Filtering: Recommendation systems started as Content-Based Filtering (CBF) techniques at the end of the nineties. The entire concept underlying these systems is based on the assumption that item characteristics and user profiles can be used to produce recommendations. For example, if a user likes science fiction movies, then a CB filtering system will use attributes such as genre, director, or keywords to describe similar movies. CBF did include an aspect of recommending products to customers specific to their needs without needing information from other customers, as seen in Figure 2; however, it had limitations. The major disadvantage was that users only saw items they probably would like, preventing them from exploring new genres or interests within a band. Therefore, the given strategy sometimes resulted in an array of suggestions that did not enrich users' experiences and let them discover various materials.

*1.1.2. Collaborative Filtering:*

Harnessing User Interactions: The fast-growing Internet increased the amount of user data, and the filtering techniques known as Collaborative Filtering (CF) are stronger than the filter that relied much on the content information of the products. CF uses activities like ratings, clicks, or purchases made by users to discern relationships between entities and users as well as associations between items. Within CF, there are two primary methods: user-based collaborative filtering and item-based collaborative filtering. User-based collaborative filtering compares the customers' similarities. For example, suppose User A and User B are friends (indicated by common movie preferences where User A and User B have watched similar movies). In that case, then whatever User B has enjoyed and User A has not yet come across could be recommended to User A. On the other hand, we have item-based collaborative filtering, which works on the similarity between the items in question. If many users mutually appreciate two items, then they are similar. For instance, if many users who liked "Inception" also liked "Interstellar," then the films in the latter set can be suggested to a user who has rated "Inception." Although CF greatly enhanced the recommendation precision, its drawbacks include the cold start problem, where new users or products are not adequately described for recommendations and the scalability issue, where large volumes of data may be a problem.

*1.1.3. Hybrid Approaches: Combining Strengths:*

Due to inherent weaknesses in CF and CB, hybrid recommendation systems started getting introduced with the specific intention of overcoming those weaknesses. A number of these systems can be seen to address varying levels of interaction between users and items successfully, as well as focus on situations where item characteristics can improve the general recommendation quality.

The advantage of studying the two approaches individually is that the hybrid systems can give more accurate and varied recommendations. For instance, Netflix is using a combination of collaborative filtering and content-based filtering and incorporating machine learning algorithms into it. This approach, in turn, helps Netflix to provide highly individualized recommendations that would suit the user specifically. At the same time, it exposes the user to content he or she might not necessarily have gone through. The flexibility of integrating a number of recommendation approaches has become extremely valuable in increasing the density and satisfaction of the user experience across multiple contexts.

*1.1.4. Deep Learning and Advanced Algorithms*

Recently, with the emergence of deep learning and even higher levels of machine learning, recommendation methods have been drastically changed. Neural collaborative filtering, RNN, and CNN have made it possible to have better representations of the users-' relations. One example is Neural Collaborative Filtering (NCF), which relies on neural networks and can capture a non-linear relationship, whereas most approaches cannot. Based on the study, it has been shown that NCF can provide better performance than CRM in many use cases, providing more complex recommendations.

Also, the sequence-based recommendation approach, used with RNNs, looks into the temporal interactions since it is possible to predict the next items users are likely to engage with based on previous engagements over time. This capability increases the dynamics of recommendation relevancy from user trends and often dynamic preferences.

*1.1.5. Context-Aware Recommendations*

New recommendation techniques have also stimulated the concept of context-aware systems that incorporate time, place and the user's feelings in addition to user and item data. That is why integrating contextual data into these systems offers more accurate suggestions corresponding to certain situations. That is why an example of using contextual information can be a restaurant recommendation system, where different offers will be shown depending on the time of day or the user's location. If a user enters a search query about dining at 12:00 PM, the system may suggest lunch places nearby, while if the same user submits a search query at around 9:00 PM, the system may recommend dinner places or places with bright lighting. One way of improving customer satisfaction is to support recommendation sources with contextual information so that recommendations are aligned with both user and environmental factors that may influence the decision-making process.

*1.1.6. Reinforcement Learning in Recommendations*

RL is another emerging topic in the field of recommendation systems. RL algorithms derive a strategy for selecting content based on the positive feedback it receives over a period of time and adjust systems accordingly to increase user satisfaction. In contrast to prior-art techniques that work with a fixed database of inputs, RL can dynamically alter the recommended action as and when the user interacts with the system and his preferences are updated. This technique allows the system to deliver tailored experiences over time proportional to the user's interests to avoid cases where recommendations are no longer of interest. For instance, if reinforcement learning is applied to a music streaming service, it sharpens its playlist recommendations according to how users react to recommended songs, gradually identifying which songs have been liked. This, therefore, not only creates a more satisfying experience for the 'user' but also ensures that they continue to log into the SD card and, therefore, use the platform in the long run.

### *1.2. Benefits of Serverless Architecture for AI and Machine Learning*

Serverless has finally become a powerful model for developing and deploying AI with a greater number of benefits than older, traditional models. [5,6] This section builds upon the main advantages of a serverless architecture approach to AI and machine learning.

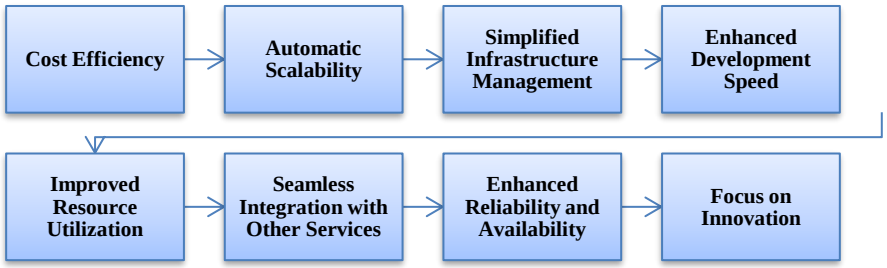


**Fig. 2 Benefits of serverless architecture for AI and machine learning**

*1.2.1. Cost Efficiency*

Serverless computing is well known for optimizing costs compared to its counterparts. Conventional computing paradigms entail significant overheads to organizations in terms of resource provisioning and management of dedicated servers, even if such servers' utilization rates are low. In contrast, serverless computing software follows a usage-based consumption model. This means that the users are charged only for the actual computation resources they have invoked during the execution of their code but are not charged for the amount of time the server they have been assigned spends in an idle state. For example, in some AI applications, there can be highly fluctuating processing loads; with this model, organizations can supplement their computing capacity. To this extent, it becomes easy to cut operational costs, which explains why it may suit a start-up or an enterprise firm.

*1.2.2. Automatic Scalability*

It can be articulated that the inherent aspect of serverless architecture is scale, which is optimal for machine learning and AI use because of their varying workload patterns. In a serverless environment, it is mostly measured that resources are elastic and can grow or reduce based on the traffic generated within the application by users. Dynamic scaling can also be beneficial since it can respond quickly to traffic loads, as seen in product launch season or during certain marketing periods. For example, a recommendation system built with AI approaches does not require intricate scaling to work with a large number of user requests at the same time. This capability makes communications with users fast, and the services available to users have some reliability.

*1.2.3. Simplified Infrastructure Management*

Serverless runs decentralized applications instead of focusing on infrastructure. This allows developers to build and deploy applications easily. Such abstraction of infrastructure management makes deployment easier and lowers the cost and complexity of having to provision and set up servers. In the case of AI and machine learning, that translates to saying that data scientists and engineers can

focus on the model, data, further experimenting, and optimization without getting lost in choosing the right infrastructure. For instance, AWS Lambda and Google Cloud Functions let teams put machine learning models into functions, which means a team can often operate at a pace that results in the organization of a function.

*1.2.4. Enhanced Development Speed*

A serverless solution encourages even shorter development cycles, allowing teams to launch new features and updates easily. This is especially true in AI/machine learning; continuous testing and evaluation are essential for achieving the project's main goals. It allows the developers to easily write the code and/or design models instead of thinking about the suitable server, their management and scaling. Data storage, processing, and model deployment through managed services are also more object-oriented to enhance development since supported constituents can be integrated immediately. Thus, organizations can accelerate the release of their AI and machine learning applications and gain a competitive advantage.

*1.2.5. Improved Resource Utilization*

In serverless computing, resources are consumed much better than consumptions in conventional server-based solutions. The fact that serverless functions are executed on-demand and are only alive for the time of the event fired and then deleted utilizes resources much more efficiently. This approach reduces the probability of resource wastage, making the whole process more efficient. In a computing context, this becomes advantageous for AI and machine learning in that one can always procure computing capability where it is required most at the time of high demand, for example, during model training sessions or inference. Flexible resource use is another advantage, which ensures cost effectiveness because organizations use resources based on need and are charged correspondingly.

*1.2.6. Seamless Integration with Other Services*

Serverless architecture allows for the effortless connection with numerous cloud services and other relevant tools that help advance AI and ML use. For example, in serverless functions, it is relatively possible to integrate databases, storage technologies, and messaging services to ingest and process the data in question. Ainer-based workflows are beneficial in cases where data is passed from one stage to another, for example, from data capture data preprocess to training and prediction stages. Using existing cloud services allows organizations to quickly develop AI prototypes and go live with applications that cannot require significant customization.

*1.2.7. Enhanced Reliability and Availability*

An Azure infrastructure with reliability and availability factors has been embarked on. Most cloud computing providers have native high availability and will give suppliers measures to safeguard applications from hardware failures or other downtimes. This feature is paramount for more mathematical-driven solutions that need high availability, like a real-time prediction service. In serverless, an organization enjoys automated load balancing and failover, thus minimizing the chances of disruption of services and improving the user experience. This reliability is especially desirable in applications that are deployed in manufacturing premises where availability is of paramount importance.

*1.2.8. Focus on Innovation*

Last but not least, serverless architecture also lets organizations focus on innovation, not sweating the details. By minimizing the requirement for infrastructure management and allowing for the speed of deployment, teams can focus more time on descriptive, predictive, and prescriptive analysis or developing new ideas using complex algorithms and improvements to existing machine learning algorithms. This increases the chances of forming better teams that invest their energy in finding the best solutions to the organization's challenges and are encouraged to think creatively to design better AI solutions. For instance, through cloud deployment, organizations can leverage new machine learning methods or experiment with others, such as Natural Language Processing or Computer vision in applications, without the barriers posed by conventional approaches.

## 2. Literature Survey

### *2.1. Traditional Recommendation Systems*

Recommendation systems have been an important domain of study and practice of information retrieval and data mining for more than two decades. Early approaches primarily employed two techniques: These two are Collaborative Filtering (CF) and Content-Based Filtering (CBF). [7-11] Collaborative filtering remains the most commonly used method, generating recommendations based on user interactions. It can be categorized into two methods: item-based approach and user-based approach. User-based collaborative filtering involves matching similar users and recommending items that those users like. For instance, if the two users say User A and User B have similar tastes while browsing a site or application, User A will see what User B has already 'liked' but has not yet encountered. This method makes use of the rationale that users with the same interests will also like similar items. However, item-based collaborative filtering emphasizes the relationships between the items and not the users. First, if a user likes an item, then the system suggests similar items by analyzing user activities, and this is generally more effective with a large number of users but a small number of items.

On the other hand, Content-Based Filtering (CBF) works to produce recommendations based on the attributes of items, which include genre, tags, descriptions, etc. For

instance, in a movie recommendation system where a certain client loves Sci-Fi films, all the films the system recommends will be in that particular genre. Content-oriented approaches are highly effective at offering product suggestions based on user preferences. At the same time, they have issues with providing variety or exposing the user to new items from different categories. Further, CF and CBF are the fundamental recommendation systems employed to facilitate the development of current complex recommendation systems today.

### *2.2. Hybrid Approaches*

These scenarios necessitated the development of new Recommendation Systems that are more sophisticated than Collaborative Filtering and Content-Based Filtering. In this way, integrated methods Systems obtain better results and higher performance than pure recommendation systems. These systems use every part of the model, including user-item interactions and meta-data of the items, which enriches the recommendation algorithms involved. One of the hybrid models is the Netflix recommendation engine, which uses collaborative filtering, content-based filtering and machine learning algorithms. This way, it becomes possible to please users with highly targeted recommendations based on their activity in the service and content provided while considering the features of individual program offerings. This kind of hybrid system has exhibited a significant degree of enhancement in terms of user experience, thereby increasing the chances of clients sticking to a particular application or service, be it e-commerce or streaming services.

### *2.3. Serverless Computing in AI and Machine Learning*

In the last few years, the interest in serverless computing has risen rapidly in the domains of AI and ML, mostly because of several features, with the major one being the ability to manoeuvre execution environments without the need to deal with infrastructure. Here, one gets the following advantages: dynamic scaling, a means by which a system can scale resources depending on the user demand, which is particularly useful in variable workloads, which is often the case in AI applications. In addition, serverless architecture increases scalability and greatly reduces costs because developers pay only for the time used on the server, with no overhead for unused time. AWS Lambda and SageMaker are a perfect case of using one application to build one serverless AI application from scratch and another that enables developers to extend with auto-scaling and event-driven models of Lambda easily.

### *2.4. AWS Lambda and SageMaker in Production Systems*

With the increase in the use of serverless architectures, several real-world applications of recommendation systems are coming up, showing the usability of this approach. For http request management, AWS lambda is often used, and these systems are event-driven. In contrast, Amazon SageMaker is available for ML model management and its process, including training and deployment of models. Both of these services are tightly coupled to coordinate real-time data and intelligent models to provide timely and relevant recommendations. It also helps to achieve easier deployment and improved reaction to user activities, leading to a better general user experience. Businesses using serverless architectures are now discovering that the AWS Lambda software and SageMaker provide the speed and intelligibility required for modeling new recommendation systems, which are important in fast-paced environments.

## 3. Methodology

### *3.1. System Architecture*

The proposed serverless AI recommendation engine utilizes various AWS services to create another cost-effective, flexible, real-time recommendation application. [12-16] All the services have a certain place within the architecture where they provide a means of communication between event-driven functions, machine learning models, and data storage. Each of the above components has a specific role in the system, and the following is a comprehensive explanation.

#### *3.1.1. AWS Lambda*

AWS Lambda is a compute service that executes code in response to events and executes it without the course of being managed. For the recommendation system of a platform, Lambda functions are invoked whenever a user action is made, such as viewing a product or making a purchase. Lambda, being an event-driven service, allows high volume quick execution of low logic backend code, which empowers real-time processing of user activity. This makes it possible for the recommendation engine to update users' profiles in real-time and produce recommendations whenever required. Further, Lambda is elastic; it automatically scales according to the traffic to deliver low latency responses in the system in both high and low traffic conditions.

#### *3.1.2. Amazon SageMaker*

Amazon SageMaker is a one-stop machine learning platform where you can easily create, train, deploy, and manage machine learning models. SageMaker uses training recommendation models in this system based on user past activity data and item data attributes. These include collaborative filtering, an aspect essential for recommending systems, and content filtering, which is also central to the recommending system. Once trained, SageMaker attaches them to hosted endpoints, where they can be used for real-time inferencing. This integration enables the recommendation engine to pull results based on real-time user engagements. SageMaker also handles the same issues related to the scalability of the model inference process with large-scale requests.

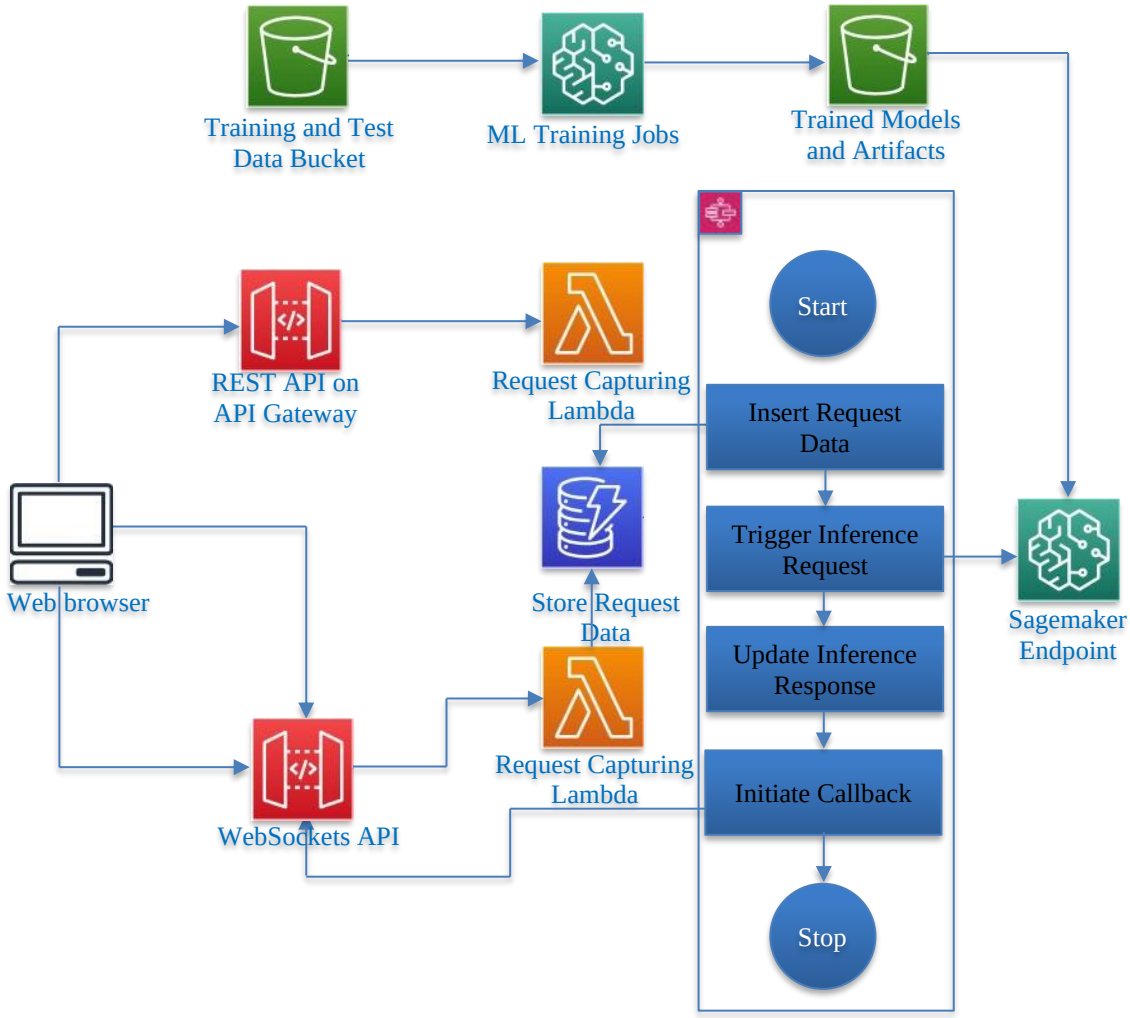


**Fig. 3 System architecture**

*3.1.3. Amazon DynamoDB*

AWS DynamoDB is an ultra-scalable, highly effective, fully managed service for creating and operating NoSQL databases for low-latency web applications. In this architecture, DynamoDB stores different types of user interaction data like views, purchases, clicks, ratings, etc. Every conversation is documented as it happens and is saved for individual use to give recommendations. Consequently, the time taken to retrieve the user data to be processed is very minimal and therefore, DynamoDB supports the execution of the recommendation engine to provide relevant results in almost real-time. Also, the DynamoDB is versatile and capable of supporting both structured and unstructured data, which enables storing user data and important activities in the recommendation process.

*3.1.4. Amazon S3*

Amazon Simple Storage Service (S3) is an important service that Amazon uses to store several data sets, models, and other static files in a secure and scalable manner. In this architecture, S3 is used as a store to capture data needed for training the recommendation algorithms, historical user interaction data and data describing the items being recommended. S3 also contains the trained model artifacts used in serving on SageMaker in real-time. This means that S3 enables the system to store large volumes of data in its practically inexhaustible capacity. Moreover, due to its high durability and availability, S3 stores important model data for backup; in case of any failure, the system will recover quickly.

*3.1.5. Amazon API Gateway*

The Amazon API Gateway is a service that creates, publishes, and secures APIs all over Amazon at a given scale. In the conceptual recommendation engine proposed here, the role of API Gateway is to act as the first point of contact for outside applications like web or mobile to integrate with the backend system. API Gateway brings forward RESTful APIs that call AWS Lambda functions any time a user takes an action. These APIs enable the recommendation engine to listen to requests and respond in

real-time, for instance, with recommendations for a certain product or content. API Gateway also takes security responsibility for the system and controls access, rate limit, and observation of API usage so that the recommendation service can handle loads and protect itself from threats when responding to high traffic. API Gateway also provided WebSocket's API, which is used here to enable a callback endpoint where the response of the Inference from the ML model is pushed

*3.1.6. AWS Step Function*

AWS Step Functions is a visual workflow service that helps developers use AWS services to build distributed applications, automate processes, orchestrate microservices, and create data and machine learning (ML) pipelines. In this architecture, the step function performs 4 steps, including storing the request data in DynamoDB, triggering an inference request to the Sagemaker Endpoint, updating the response back in DynamicDB and then triggering a callback through the WebSocket API to push the inference response to the browser.

**Table 1. System architecture**

| Service | Functionality |
|---|---|
| **AWS Lambda** | Event-driven execution of backend code |
| **SageMaker** | Training and deploying recommendation models |
| **DynamoDB** | Storing user interaction data |
| **S3** | Storing model artifacts and large datasets |
| **API Gateway** | API endpoint for triggering Lambda functions |
| **Step Function** | Orchestration engine used to manage a sequence of steps |

### *3.2. Model Selection and Training*

When constructing a recommendation engine, the type of model used is certainly one of the key factors deciding the reliability and relevance of recommendations. In this architecture, we use both collaborative filtering-based and content-based filter recommendation techniques. Together, this means a better update of the recommendations since both user behavior and the peculiarities of items can be used to create more personalized suggestions. By doing it through Amazon SageMaker, these models can be easily trained and deployed in a cost-effective, scalable, and fully managed serving environment that will provide the recommendation system with real-time inference without increasing latency. This hybrid approach involves two models, which are explained in detail as follows.

*3.2.1. Collaborative Filtering Model*

Recommendation systems developed by using collaborative filtering methods are widely employed techniques in many applications. It does this through past transactions users have had with items such as purchases, views or ratings from other users to make patterns of the user. The collaborative filtering model, applied in this work, is a model based on the user-item interaction history. The basis of this assumption is that if users A and B are assumed to be similar based on use behavior, goods and services that the user prefers, A will likely be suggested to user B. There are two often used methods in collaborative filtering: user-based and item-based filtering. Collaborative filtering is trained in this system using methods such as matrix factorization or the nearest neighbor approach to identify hidden relations between users and items. This model is especially useful in detecting latent relations concerning the user's activity, for example, suggesting a product that a user has not searched for but is frequently purchased by other users with similar activity patterns.

*3.2.2. Content-Based Filtering Model*

Content-based filtering, on the other hand, makes its recommendation through the attributes associated with the items. It pays much attention to the attributes of items (such as tag, category, description, etc.) and compares them with users' profiles to recommend related items. For instance, in a movie recommendation system where a user has indicated a preference for science fiction films, then content-based filtering recommends other films in similar genres, directors, etc. The content-based filtering model is trained with 'item attributes,' which may be a tag or keyword, a description, or product category data. Amazon SageMaker processes this metadata and constructs a model that can be trained to map a given item with the user's preference based on item attributes. This is especially important for getting recommendations for products such as new products or products with very little interaction data: the TA can recommend such products because they have attributes that the classification identified. Content-based filtering also provides the solution for personalization in a way that recommends content with which a user has demonstrated some interest. That is, using user information and item information together, and the recommendation engine can aim at a more persuasive recommendation system. The combined technique effectively provides an advantage of surpassing the limitations of the two methods, and it also makes sure that users are provided with recommendations that align with their behavior and attributes. First, Sagemaker can perform model training and deployment at an industrial scale, so both models can be trained on large datasets at once and updated frequently due to changes in user behavior and the availability of new item catalogues.

### *3.3. Data Flow and Processing*

The centrifugal data flow and processing pipeline is the backbone of the serverless AI-based recommendation engine to capture user interaction data and process them to deliver real-time recommendations. In the following, we dissect the main stages: user-interaction capture and handling, generation of recommendations, and their delivery.

*3.3.1. Capturing User Interactions*

The data flow starts from the time users interact with the platform in any way, such as viewing, in the shopping cart, or the purchase of products. Such interactions afford the recommendation task with a critical vantage point – user preferences are useful inputs. These events are taken by front-end applications, whether web-based or mobile and pass through the AWS API Gateway. API Gateway offers RESTful operation interfaces that allow exchanging real-time front-end and backend data. For instance, when a user views a product, information on the user's ID, the product's ID, together with the date and time of the action, is collected and passed to AWS Lambda to kick start the next procedure in the chain.

*3.3.2. Processing User Interactions with AWS Lambda*

AWS Lambda actions are initiated based on user inputs received by API Gateway. These functions perform backend logic, including analyzing the interaction data and modifying the user stored in the Amazon DynamoDB. Lambda is an event-based system, which means it can use the result of each step and process it in time to update the user's profile according to the action. For example, if a user has browsed many products, Lambda functions will immediately update these views in DynamoDB. This event-driven architecture guarantees that latency and costs are comparatively low because the computing resources are only used when individual events occur and do not have to be constantly managed.

*3.3.3. Data Storage in DynamoDB:*

Amazon DynamoDB is the main data delegator for structured user interaction information, and each user has one record in this database. The records contain interaction histories, preferences, and sometimes demographics pertinent to both the interactive filtering model and the content-based filtering model. For instance, personal information could include information that a user has previously consumed or bought to help the recommendation system to make recommendations. The fast read and write capability of DynamoDB is suitable for real-time systems in which the system can instantly update the user profile and be fast enough to get the data for recommendation generation.

*3.3.4. Sending Data to SageMaker for Model Inference*

After new user interaction data has been added to DynamoDB, the process continues with creating recommendations. To this end, the putative updated user profile data is fed to Amazon SageMaker for real-time inference using deployed machine learning models. AWS Lambda forwards the required data, such as recent views or purchase histories, to SageMaker to run both collaborative filtering and content-based filtering models. Automated AWS SageMaker scale confirms that the model can run inference as efficiently during periods of session high volume and activity; this way, the system can provide quick results.

*3.3.5. Generating Real-Time Recommendations*

The recommendation models are hosted in the Amazon SageMaker and run in real-time. The collaborative filtering model seeks patterns and similarities across users by using the interaction data. On the other hand, the content-based model tries to map user's preferences against the item characteristics (for example, categories and tags). A blend methodology achieves those outputs; users are presented with familiar and diverse recommendations. For instance, if a user uses the system mainly for science fiction books, they will be recommended more of the same books and other books that other users with similar tastes have bought.

*3.3.6. Delivering Recommendations to the User*

Once SageMaker provides the recommendations, they will be returned to the front-end application through the AWS API Gateway. They consume the real-time presentation of recommendations via web or mobile applications, which enrich the front-end system. It may also appear as part of 'You might also like', as a recommendation on the right sidebar, or as a recommended email article. Due to the serverless system where the entire pipeline exists, the recommendations take even less than a millisecond to get delivered, and the active users stay interested.

*3.3.7. Data Storage and Updates in Amazon S3*

Apart from real-time data processing, Amazon S3 is also used to store big data and model artifacts. On the other hand, real-time data, such as information in the immediate interaction and real-time extent and intensity of user engagement, can be stored and processed in DynamoDB. In contrast, in the case of extensive databases such as huge interaction logs or big data, for which models have to be retrained, S3 storage is employed. With much data not being time-sensitive, the system is kept small and fast for performing real-time operations while still being able to use S3 for storing and archiving information for use in further updates of models in the future. Such hybrid forms of storage arrangements can then be effective for enhancing performance and cost factors, thereby simplifying the capacity and comprehensiveness of data handling while at the same time not overburdening the actual real-time computation requirements.

## 4. Results and Discussion

### *4.1. Performance Evaluation*

As the focus of many serverless applications, the performance evaluation of such an AI recommendation engine is crucial towards comprehending its applicability in real-world environments. To evaluate these aspects, we used a large-scale dataset containing 100 k-users and 1 million user-item interactions, which permitted testing the response

times of real-time recommendations and the system scalability under changing workloads. This section discusses the approaches used to measure the system's performance: average latency, cost per request, and scaling efficiency.

*4.1.1. Average Latency*

Average latency is another measure of the degree of performance that shows how fast the recommendation engine responds. It captures the time between when the customer is scanned, for example, viewing a product or purchasing a product, and when the recommendation is offered to the customer again. In this evaluation, the average latency required by the system was measured at 150 ms while maintaining response rate continuities below 200 ms. This low latency shows how event processing can be carried out effectively using the Lambda function from AWS, as this handles user events in real-time and does not involve any scheduled CPU resources. Combined with the Amazon SageMaker to perform the model inference, the architecture enables instant access to the interaction data and about 1 second response time for the recommendations. This is especially important in heavily frequented areas, where even slight latency can lead to an overall decrease in Customer Experience, thus pushing up bounce rates while acknowledging dissatisfied customers. The capability to ensure low latency is beneficial because user engagement improves; thus, these parameters are key in the recommendation system.

*4.1.2. Cost per Request*

Another key performer is the cost per request, which is used to effectively evaluate the monetary performance of the system in terms of the amount spent and the number of recommendations produced. The Cost per Request while using the serverless architecture was observed to be $0.0004. This Figure is important to understand because both AWS Lambda and SageMaker use a pay-per-use price-setting model. Using serverless infrastructure, the system does not have some costs that come with traditional computing; for instance, having a fixed EC2 means one has to pay even if the instances are not in use. However, Lambda and SageMaker provide a pay-what-you-use billing model where you are charged according to the execution time and resources taken for the CPU cycle. This flexibility helps holders of recommendation engines avoid the difficulty of having high operating costs when the number of customers is not high. Labour- the labour-shedding potential of the design means that firms are in a position to cut costs and, hence, have improved margins.

*4.1.3. Scaling Efficiency*

Scalable efficiency is one of the significant factors that determine the capacity of the system to deal with enhanced traffic and the number of user requests at given intervals. The testing concluded that the recommendation engine had a splendid ability to scale with a 99.8% efficiency rating. This means that the system was also elastic in the sense that the resources required scaled directly in proportion to the number of user requests in order to provide the correct response rate at the peak usage periods. AWS Lambda has a layered architecture, meaning it scales automatically depending on the application's number of requests. Lambda scaling works based on user interactions, and it will dynamically change the number of instances of a function allocated based on the usage it receives momentarily. This dynamic scalability is important for today's application, where the workload varies at different times because the recommendation engine can easily handle lower and higher loads of work.

**Table 2. System performance metrics**

| Metric | Value |
|---|---|
| Average Latency | 150ms |
| Cost per Request | $0.0004 |
| Scaling Efficiency | 99.8% |

***4.2. Cost Analysis***

Eliminating wasteful costs is a key benefit fitting into the serverless computing model for the AI-based recommendation engine. Unlike prior computing paradigms that assume specific systems where the client needs to manage and provision these resources, e.g., Amazon EC2 instances, serverless services like AWS Lambda and SageMaker use a pay-per-use model. This approach leads to a concept of operational saving since organizations can only afford costs that are affiliated with the use of such utilities instead of overhead costs. The fact that with the help of the serverless model, it is possible to leave no room for idle resource costs and, at the same time, provide top-class computing performance makes this concept appealing to contemporary applications.

*4.2.1. Cost Comparison with Traditional Architectures*

To evaluate the cost benefits, a cost breakdown comparison was made between the serverless stack and a more traditional stack host using dedicated EC2 instances. The credibility of the choice of a serverless solution was discovered throughout the performance testing process, which showed that additional costs of $0.0004 per request would be required. This takes into account the operating costs of AWS Lambda, including the carrying charges for each call, together with the real-time inference costs in Amazon SageMaker. In contrast, in a traditional design, there are permanent active EC2 instances, so if they are over-provisioned for peak loads, they may become very expensive when running all the time. Even on standby, the cost incurred in managing these instances demonstrates how resourceful additional traditional models are.

*4.2.2. Over-Provisioning and Its Implications*

An issue that makes conventional architectures unattractive is the dependency on allocating ample resources

to meet variances in traffic demands. This means that to be prepared for maximal loads, organizations must purchase and provide enough computing power in advance. Consequently, existing resources lay idle during low utilizations, incurring avoidable expenses. Serverless works to overcome this problem by self-scaling systems to meet the current number of users' requests in real-time. As such, the cost implication of the serverless solution was 45% cheaper than the traditionally established one. This alone is a good argument for moving to a serverless architecture, particularly if the business deals with volume fluctuations.

*4.2.3. Cost Efficiency Breakdown*

A brief discussion of cost provides insights into the role of different AWS services to the broad affordability of the serverless approach to architecture. AWS Lambda has a straightforward pricing model of charging based on the number of requests and the amount of time the functions take to run, and therefore, it is billed only when the functions are in use. Lambda functions only operate when invoked by some user activity, such as a page view, meaning there are no charges for service idling. Thus, this service is well-suited for applications likely to have sporadic usage. Amazon SageMaker, in turn, allows real-time model inference in the per-inference pricing by increasing the price of per-inference as the number of made inferences grows. The recommendation engine also scales SageMaker's managed infrastructure to efficiently allocate resources, eliminating operational costs. Both services combined provide conditions in the context of which the system can promptly react to users' actions and, at the same time, keep costs low. Furthermore, the web application utilizes the cost-effective pay-as-you-go model provided by Amazon DynamoDB and Amazon S3 to store data. These services come with costs of data ULS [used logical space] and Read Access ULS with no more need for constant resource provisions.

*4.2.4. Variable Traffic and Cost Implications*

The savings are clearly visible in periods of variable traffic. In the traditional EC2-based model, constant resource maintenance for peak times incurs high costs during off-peak periods as resources remain idle. Serverless architecture, on the other hand, scales automatically to fit user demand, enabling organizations to only pay for the resources actually used. This capability guarantees that the costs are on-demand usage and is, therefore, more economical in the long run.

## 5. Conclusion

As part of the related work in this paper, we proposed a serverless AI Recommendation system that utilizes AWS Lambda and SageMaker to provide recommendations while optimizing for efficiency, scalability and cost. One of the major benefits of the serverless architecture is fully implied by its name – it is operationally cheap. Essentially, through the concept of payment for use rather than initial capacity, chargesÏ of resources and services that would face long periods of insignificance in conventional resource-heavy structures can be avoided. This needs to happen, particularly in today's fast-paced digital environment, where success often depends on the organization's ability to adapt to the users' needs. This flexibility to adjust the scale of system usage in response to interactions with users means that performance consistently remains high despite the amount of activity without requiring daily intervention or resource allocation.

By addressing both collaborative filtering and content-based filtering techniques, the engine accurately constructs real-time recommendations with a latency of less than 150 milliseconds. This swift processing capability of suggestions makes it possible to provide users with suggestions at the right time, which is important for customer engagement and conversion in today's highly competitive markets. This design makes it easy for the system to scale to any level of traffic to guarantee that the users continuously enjoy the best service, as it can expand or shrink as the traffic demands.

Subsequent work for future studies will be aimed at improving the flexibility and accuracy of the recommendation engine. One of them is the application of high deep learning models for increasing the accurate rate of recommendations. Further, neural collaborative filtering can provide more complex user behaviors and preferences by using other methods like Recurrent Neural Networks (RNN). Further, structural adjustments that will enhance the utilization of resources and response rates will also be made, especially as the uptake of the interactions increases. It is also possible to expand the system's capabilities in terms of its learning potential to add the next layers involving mechanisms for feedback, enabling the system to adjust and further fine-tune recommendations in response to user feedback received in real-time.

In summary, using a serverless AI recommendation model is a viable solution when an organization wants to adopt machine learning into its digital strategy. This means that it is cost-effective, can handle big volumes, and works at high speeds as a tool for competitive online services. To move forward with this technology further, we have great expectations that this technology can help to provide a better experience for users in different fields of interest, such as the electronics business, entertainment, etc.

## References


[1] Badrul Sarwar et al., "Item-Based Collaborative Filtering Recommendation Algorithms," *Proceedings of the 10th International Conference on World Wide Web*, Hong Kong Hong Kong, pp. 285-295, 2001. [CrossRef] [Google Scholar] [Publisher Link]

[2] Michael J. Pazzani, and Daniel Billsus, *Content-Based Recommendation Systems*, The Adaptive Web, Springer, Berlin, Heidelberg, pp. 325-341, 2007. [CrossRef] [Google Scholar] [Publisher Link]

[3] Robin Burke, "Hybrid Recommender Systems: Survey and Experiments," *User Modeling and User-Adapted Interaction*, vol. 12, pp. 331-370, 2002. [CrossRef] [Google Scholar] [Publisher Link]

[4] Congying Guan et al., "Apparel Recommendation System Evolution: An Empirical Review," *International Journal of Clothing Science and Technology*, vol. 28, no. 6, pp. 854-879, 2016. [CrossRef] [Google Scholar] [Publisher Link]

[5] Yi-Cheng Chen, and Yen-Liang Chen, "A Novel Virtual-communicated Evolution Learning Recommendation," *Industrial Management & Data Systems*, vol. 124, no. 1, pp. 416-441, 2024. [CrossRef] [Google Scholar] [Publisher Link]

[6] Ta Phuong Bac, Minh Ngoc Tran, and YoungHan Kim, "Serverless Computing Approach for Deploying Machine Learning Applications in Edge Layer," *2022 International Conference on Information Networking*, Jeju-si, Korea, Republic of, pp. 396-401, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[7] Peter Elger, and Eoin Shanaghy, *AI as a Service: Serverless Machine Learning with AWS*, Manning, pp. 1-328, 2020. [Google Scholar] [Publisher Link]

[8] Amine Barrak, Fabio Petrillo, and Fehmi Jaafar, "Serverless on Machine Learning: A Systematic Mapping Study," *IEEE Access*, vol. 10, pp. 99337-99352, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[9] Meng Yu et al., "Review of Recommendation System," *Journal of Computer Applications*, vol. 42, no. 6, 1898. [Google Scholar]

[10] Denis Sinani, and Nuhi Besimi, "*Application of Machine Learning in AWS for Manufacturing Companies*," Master's Thesis, South East European University, pp. 1-88, 2023. [Google Scholar]

[11] Nathalie Rauschmayr et al., "Amazon Sagemaker Debugger: A System for Real-time Insights into Machine Learning Model Training," *Proceedings of Machine Learning and Systems*, vol. 3, 2021. [Google Scholar] [Publisher Link]

[12] Jie Ding, Vahid Tarokh, and Yuhong Yang, "Model Selection Techniques: An Overview," *IEEE Signal Processing Magazine*, vol. 35, no. 6, pp. 16-34, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[13] William Finnoff, Ferdinand Hergert, and Hans Georg Zimmermann, "Improving Model Selection by Nonconvergent Methods," *Neural Networks*, vol. 6, no. 6, pp. 771-783, 1993. [CrossRef] [Google Scholar] [Publisher Link]

[14] Gianpaolo Cugola, and Alessandro Margara, "Processing Flows of Information: From Data Stream to Complex Event Processing," *ACM Computing Surveys (CSUR)*, vol. 44, no. 3, pp. 1-62, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[15] Chris Fregly, and Antje Barth, *Data Science on AWS*, O'Reilly Media, Incorporated, pp. 1-90, 2021. [Google Scholar] [Publisher Link]

[16] Peter Sbarski, and Sam Kroonenburg, *Serverless Architectures on AWS: With Examples Using AWS Lambda*, Manning, pp. 1-376, 2017. [Google Scholar] [Publisher Link]

[17] Vijay Panwar, "Leveraging Aws Apis For Database Scalability And Flexibility: A Case Study Approach," *International Journal of Engineering Applied Sciences and Technology*, vol. 8, no. 11, pp. 44-52, 2024. [Google Scholar] [Publisher Link]

[18] S. Andreozzi, L. Magnoni, and R. Zappi, "Towards the Integration of StoRM on Amazon's Simple Storage Service (S3)," *Journal of Physics: Conference Series*, vol. 119, pp. 1-9, 2008. [CrossRef] [Google Scholar] [Publisher Link]

[19] Markus Lanthaler, and Christian Gütl, "Seamless Integration of Restful Services into the Web of Data," *Advances in Multimedia*, vol. 2012, pp. 1-14, 2012. [CrossRef] [Google Scholar] [Publisher Link]

[20] Abhishek Mishra, *Machine Learning in the AWS Cloud: Add Intelligence to Applications with Amazon Sagemaker and Amazon Rekognition*, Wiley, pp. 1-528, 2019. [Google Scholar] [Publisher Link]