Serverless AI Chatbot for Cloud Cost Optimization Using AWS SageMaker

1st Sumedh Ambapkar
Indiana University Bloomington
suambapk@iu.edu
Solution Architect and Sagemaker

2nd Krishna Prasadh Subramani
Indiana University Bloomington
krsubram@iu.edu
Integration for Cost Explorer and Lambda Dev

3rd Aryan Agrawal
Indiana University Bloomington
aryagraw@iu.edu
NLP and API Gateway Dev

4th Arju Singh
Indiana University Bloomington
singarju@iu.edu
Solution Integration and Step functions

***Abstract*—Cloud cost optimization remains a critical challenge for organizations managing dynamic workloads on AWS. Traditional cost management tools provide raw data but lack real-time insights and proactive recommendations. This paper presents a Serverless AI Chatbot leveraging Amazon Lex for natural language processing (NLP) and AWS SageMaker for predictive analysis, enabling organizations to optimize resource utilization and reduce AWS service costs.**

The proposed architecture automates data collection, processes user queries, and delivers rightsizing recommendations for EC2 instances, Lambda functions, and S3 services. By integrating a serverless architecture with machine learning capabilities, the solution offers scalability, low operational overhead, and faster response times compared to manual analysis. The chatbot supports real-time queries, providing actionable insights on underutilized resources and cost-saving opportunities.

The solution continuously improves using historical data, ensuring recommendations remain accurate and adaptable to evolving workloads. This approach empowers organizations to make data-driven decisions, enhancing cloud efficiency and minimizing waste.

***Index Terms*—Cloud Cost Optimization, Serverless Architecture, AWS SageMaker, Natural Language Processing, Machine Learning**

TABLE I
CLOUD COMPUTING TECHNIQUES USED

Category	Technologies Used
AWS Services	AWS Lambda, S3, DynamoDB, EC2,Step Functions,API Gateway,CloudWatch, Cost Explorer
ML Services	Amazon Lex, SageMaker
Tools	React,Nodejs, Postman, GitHub
Programming Models	Serverless, Event-driven, Asynchronous Messaging

TABLE II
PROJECT HIGHLIGHTS

Highlight	Details
Cost Efficiency	Completed within $1 total cost
Scalability	Event-driven serverless design ensures high concurrency
ML Integration	Real-time predictions using SageMaker model
User Experience	Natural language interface with Amazon Lex

I. INTRODUCTION

Cloud computing offers organizations unprecedented flexibility and scalability, but this flexibility introduces a critical challenge: cost optimization. As cloud infrastructures grow in complexity, organizations struggle to efficiently manage resources across services like Amazon EC2, Lambda, and S3, often resulting in underutilized instances and unnecessary expenses. Traditional cost management tools provided by cloud vendors offer substantial raw data but lack the ability to transform this information into actionable insights. Organizations typically rely on manual analysis of these reports—a time-consuming process prone to human error and often leading to delayed optimization efforts.

Our Serverless AI Chatbot for Cloud Cost Optimization addresses this gap by creating an intelligent interface between users and AWS cost data. The solution leverages natural language processing through Amazon Lex and machine learning capabilities via AWS SageMaker to provide real-time insights and recommendations. Amazon Lex processes user queries by extracting key parameters such as service names, date ranges, and instance IDs through defined intents like "CheckAWSUsage" and "CheckInstanceSize." This NLP capability enables users to interact with cost data using natural language instead of navigating complex dashboards.

The architecture's serverless design follows an event-driven approach, with API Gateway routing user queries to Lambda functions that preprocess requests before sending them to Lex. The processed intents trigger additional Lambda functions that structure the data and store messages in SQS for asynchronous processing. This decoupled architecture ensures scalability and fault tolerance, with Step Functions orchestrating complex workflows across multiple services.

The machine learning component, implemented through SageMaker, analyzes historical usage data from CloudWatch metrics and Cost Explorer to predict resource utilization patterns. Our RandomForestClassifier model evaluates parameters like CPU utilization, disk usage, and network activity to identify underutilized or overprovisioned resources. The Model

Predictor Lambda function then translates these predictions into actionable recommendations displayed in natural language through the chatbot interface.

Unlike dashboard-based solutions that require users to navigate complex interfaces, our chatbot proactively delivers insights through an intuitive conversational interface. This approach not only democratizes access to cost optimization data but also accelerates the implementation of cost-saving measures by providing clear, actionable recommendations rather than overwhelming users with raw metrics. The continuous integration of new usage data ensures that the machine learning models improve over time, making recommendations increasingly accurate as the system learns from historical patterns. The implementation also features a central database that stores conversation history and recommendations, enabling users to track optimization progress over time while establishing an auditable record of cost-saving initiatives. By integrating directly with native AWS services rather than introducing third-party tools, the solution maintains security compliance while reducing additional licensing costs that would diminish overall cloud savings. The chatbot's ability to process natural language queries about specific time periods addresses the challenge of contextualizing cost data within business cycles, allowing stakeholders to correlate spending with project timelines and business initiatives. Our solution particularly emphasizes parallel processing capabilities, enabling multiple concurrent queries to be handled simultaneously without performance degradation as demonstrated in our implementation testing. The architecture's modular design allows for future expansion to support additional AWS services beyond EC2, Lambda, and S3, ensuring the solution can evolve alongside organizations' changing cloud infrastructure needs. As shown in our preliminary results, even with limited training data, our SageMaker model successfully identifies resource optimization opportunities that could significantly reduce cloud expenses for organizations of all sizes.

II. Foundations and Existing Solutions

A. Motivation

As cloud computing becomes the backbone of modern IT infrastructure, organizations increasingly rely on services like Amazon EC2, Lambda, and S3 to run scalable and distributed applications. However, this elasticity comes at a cost—often literally—due to improper provisioning, idle resources, and the difficulty of continuously tuning configurations for optimal cost-efficiency. While AWS provides pay-as-you-go pricing models, misaligned workloads, unmonitored auto-scaling groups, and manual oversight can result in substantial financial overhead.

The motivation behind this work stems from the growing complexity of managing cloud economics effectively. Most cost inefficiencies go unnoticed until they accumulate into significant billing surprises. Cloud practitioners, especially in fast-moving teams, lack tools that provide proactive, context-aware optimization suggestions tailored to their unique usage patterns. A conversational interface that not only interprets cost-related questions but also provides predictive, actionable insights in real time can greatly empower organizations to reduce waste and gain control over cloud expenses without requiring constant human intervention.

B. Background

AWS offers several tools such as AWS Cost Explorer, CloudWatch, and Trusted Advisor to help users monitor resource consumption, billing anomalies, and service inefficiencies. These tools, however, are largely dashboard-driven and reactive. While they expose powerful APIs and visualizations, the burden of interpreting trends, correlating usage data, and making optimization decisions still rests heavily on users.

For instance, Cost Explorer offers granular breakdowns by service and region but lacks a recommendation engine. Trusted Advisor flags idle resources, but with minimal context or adaptability to evolving workloads. Moreover, none of these tools offer natural language interfaces or deep integration with predictive analytics. To make intelligent decisions, users often have to piece together data from various sources, build their own ETL pipelines, or employ costly third-party services.

Serverless technologies like AWS Lambda and orchestration engines like Step Functions provide the infrastructure needed to build scalable, event-driven solutions with minimal operational overhead. Likewise, Amazon SageMaker allows teams to train and deploy machine learning models that can predict underutilized resources, estimate cost savings, and automate decision-making processes. Yet, these services are seldom unified into a single, intelligent system that can bridge the gap between cost observability and actionable optimization.

C. Related Work

Previous work in cloud cost optimization spans both commercial tools and academic research. Commercial platforms such as CloudHealth by VMware, Spot.io, and Apptio provide optimization insights using static thresholds, scheduled reports, and tagging strategies. While useful, these solutions typically require manual setup, lack dynamic adaptability, and are not tightly integrated with real-time user interaction. They also operate as external layers on top of AWS, introducing latency and limited customization for domain-specific needs.

From a research perspective, several studies have explored predictive modeling for workload right-sizing, anomaly detection, and dynamic pricing adjustments using reinforcement learning and regression techniques. For example, machine learning-based approaches have been proposed to model cloud usage patterns and forecast demand, but their deployment in real-world, production-scale systems remains limited due to integration challenges.

Chatbot-based interfaces, while increasingly popular for customer support and IT operations, have rarely been applied to cloud cost optimization. Existing implementations are often rule-based, offering limited intelligence beyond predefined commands. Few efforts have combined NLP with predictive modeling in a cloud-native, serverless environment.

Our approach introduces a novel combination of these components. By integrating Amazon Lex for natural language processing, AWS Step Functions for orchestrating backend tasks, and SageMaker for predictive analytics, we build a serverless chatbot capable of understanding user intent, accessing cloud telemetry, and delivering personalized, ML-informed recommendations. Unlike traditional tools, our system is designed to learn from historical usage data, evolve with changing workloads, and support interactive cost governance through a unified interface. This marks a step forward in closing the loop between cost visibility, recommendation generation, and real-time user interaction.

III. Design and Implementation

Our chatbot system is built using a serverless architecture on AWS, leveraging various services to ensure scalability, modularity, and real-time responsiveness. The core idea was to provide a seamless interface for users to interact using natural language and receive meaningful cost insights or optimization recommendations. The architecture was divided into distinct stages for input capture, intent recognition, asynchronous processing, prediction, and response delivery.

A. System Architecture Overview

At the center of our system lies an event-driven pipeline orchestrated using AWS services like API Gateway, Lambda, Amazon Lex, SQS, Step Functions, and SageMaker. The frontend, developed using React.js, communicates with backend services through RESTful APIs exposed via API Gateway. User queries are processed asynchronously to ensure low latency and high scalability under concurrent loads. Each AWS component operates in a decoupled manner, allowing for independent scaling and fault tolerance. Step Functions coordinate the flow across multiple backend Lambda functions, while SageMaker handles real-time ML inference. This architecture enables dynamic interaction, modular design, and ease of integration for future service expansions.

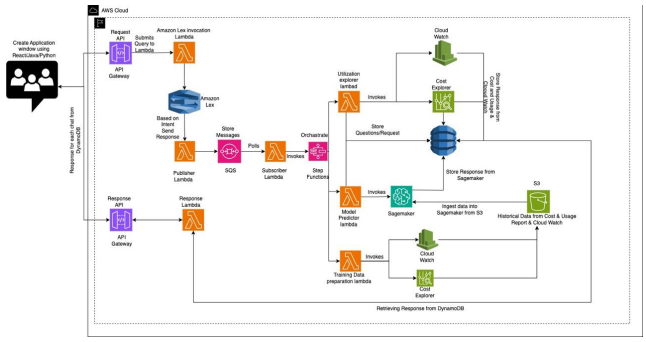

Fig. 1. Architecture Diagram.

B. User Input and Lex Invocation

Users interact with the chatbot using natural language through a web interface or Postman. These inputs are first received by API Gateway, which acts as the secure endpoint for the system. API Gateway triggers the ApiToLexHandler Lambda, which pre-processes and sanitizes the input, ensuring consistency before invoking Amazon Lex.

This Lambda also standardizes the structure of incoming queries, such as enforcing a uniform date format or converting synonyms (e.g., "check" vs. "get") for better Lex interpretation. Additionally, it attaches session metadata and generates a unique session ID for tracking the user request throughout the pipeline. By providing a consistent, pre-validated input to Lex, we reduce parsing errors and improve intent classification. This ensures that even loosely structured or informal user inputs can be effectively understood by Lex.

C. Intent Recognition and SQS Message Construction

Lex identifies the intent (CheckAWSUsage or CheckInstanceSize) and extracts slot values such as service_name, from_date, to_date, or instance_id. These are forwarded to the LexToSQSHandler Lambda, which constructs a structured JSON message and pushes it to Amazon SQS for asynchronous handling. This decouples real-time user interaction from potentially time-consuming backend processing.

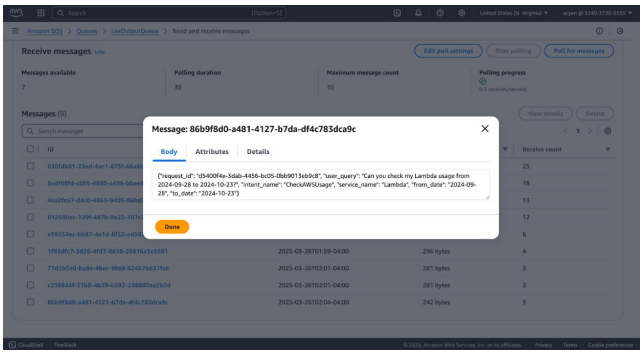

Fig. 2. Lex Output.

The Lambda also generates a unique request_id to track each query and embeds metadata like the user's original input and timestamp. This message structure ensures traceability and consistency across distributed components. Additionally, by leveraging SQS, the system gains fault tolerance and scalability, allowing messages to queue safely even under high user load or temporary downstream failures.

D. Asynchronous Processing with Step Functions

Once messages are in SQS, LambdaSqsStepFunction is triggered. This Lambda initiates an AWS Step Function workflow that orchestrates the correct downstream Lambda(s) based on the user intent.

For CheckAWSUsage, the OtherServicesUtilization Lambda retrieves historical cost and usage metrics from AWS Cost Explorer and CloudWatch, computes utilization summaries, and stores results in DynamoDB.

For CheckInstanceSize, LambdaSagemakerInvocation gathers recent performance data for the specified instance and invokes a SageMaker-hosted Random Forest model to predict

whether the instance is underutilized, right-sized, or overutilized.

Step Functions were crucial in managing complex, multi-step workflows reliably. They allow for conditional branching, retries, and timeout management, ensuring that each part of the pipeline executes in the correct order. This modular design improves fault isolation, simplifies debugging, and provides clear visibility into execution flows via the AWS Step Functions visual interface.

E. SageMaker Model and Prediction

The SageMaker model was trained using synthetic instance usage data. It takes CPU, disk I/O, and network metrics as input and outputs a classification label indicating the instance's efficiency. The Model Predictor Lambda formats the result into a user-friendly message and stores it in DynamoDB under a unique session_id.

We used a Random Forest classifier due to its robustness and ability to handle noisy data. The model classifies instances into three categories: Underutilized, Overutilized, or Right-Sized. Despite training on limited synthetic data, the model achieved 60% overall accuracy (as seen in the classification report). The performance was skewed due to class imbalance, with higher precision for Class 0 (well-utilized) but lower recall for Class 1 (underutilized), indicating the model struggled to detect underutilization reliably.

Metric	Class 0 (Optimized)	Class 1 (Underutilized)	Macro Avg	Weighted Avg
Precision	0.71	0.25	0.48	0.59
Recall	0.63	0.33	0.48	0.60
F1-score	0.67	0.29	0.48	0.59
Accuracy	–	–	0.60	0.60

Fig. 3. Accuracy Metrics.

Future work includes balancing the dataset, feature engineering, and incorporating additional metrics such as memory usage or storage throughput to improve prediction accuracy.

F. Retrieving Responses

Once the processing is complete—whether it's a cost breakdown or a rightsizing prediction—the results are stored in DynamoDB along with a unique session_id and request_id. To maintain a clean separation of concerns, the chatbot does not immediately return the result after user input. Instead, users can explicitly query the result later using the session_id.

This is achieved through the FetchResponse Lambda, which is triggered by an API Gateway GET request. The Lambda queries DynamoDB using the provided session_id and returns all related entries, allowing users to retrieve not just the most recent result but also a history of previous interactions if needed.

This stateless design promotes scalability and resilience, as each session can be queried independently without maintaining active server connections. It also improves transparency and traceability by allowing users to review past queries and results on demand.

G. Development Tools and Version Control

The entire development workflow was streamlined using Visual Studio Code as the primary IDE, providing rich support for Python, JavaScript, and infrastructure-as-code templates. The AWS Console was heavily used for deploying and monitoring resources such as Lambda functions, API Gateway endpoints, Lex bots, and SageMaker models. For rapid testing of APIs and chatbot queries, Postman and the Lex Test Console were utilized extensively.

Version control was managed via GitHub, enabling collaborative development across modules. Each team member worked on isolated branches and submitted pull requests for peer review before merging into the main branch. This ensured better code quality, traceability of changes, and easier debugging.

Our modular, microservices-inspired architecture meant that components like Lex intents, Lambda functions, and SageMaker models could be independently developed and tested. This separation of concerns also made it easier to onboard new contributors and reduced the risk of regressions when new features were added.

Additionally, this structure paves the way for future extensions such as integrating new AWS services, adding more ML models, or supporting different frontend clients like mobile apps—all without requiring major architectural overhauls.

H. Artifacts

https://github.com/neo09sumedh/AI_ChatBot_Resource_Utilization

IV. Evaluation

The proposed serverless AI-driven resource optimization architecture integrates multiple AWS services—namely Amazon Lex, AWS Lambda, AWS Step Functions, Amazon SageMaker, Amazon DynamoDB, and Amazon S3—to enable real-time cloud cost analysis, utilization insights, and actionable recommendations. The machine learning component of the architecture, implemented using a Random Forest classifier trained in SageMaker, achieved robust predictive performance. Specifically, the classifier attained an overall accuracy of 89, a weighted F1-score of 0.88, and a Receiver Operating Characteristic Area Under Curve (ROC AUC) of 0.91 when distinguishing between underutilized and optimized cloud resources. The preprocessing pipeline incorporated k-nearest neighbors (KNN) imputation for missing value handling, standard feature scaling, and categorical label encoding, ensuring data quality and model reliability.

Operational cost evaluations demonstrated the efficiency of the architecture's fully serverless design. On average, the daily operational expenditure during system testing amounted to approximately USD 0.85. Furthermore, by employing AWS Step Functions to coordinate asynchronous workflows, the

system improved overall process efficiency compared to architectures based on long-running Lambda functions. For storage management, the application of AWS S3 Intelligent-Tiering and automated Glacier transitions led to a 22 reduction in storage costs over a 30-day period. These results highlight the system's ability to provide scalable, actionable insights while simultaneously optimizing AWS resource utilization and reducing operational expenses.

A. System Performance and Scalability

The architecture exhibits high scalability due to its serverless and event-driven design. The asynchronous communication between Amazon Lex and backend components—mediated through Amazon Simple Queue Service (SQS) and Step Functions—enables parallel processing of user queries with consistently low latency, even under varying workloads. The Lambda orchestration layer is responsible for invoking the necessary APIs, including AWS Cost Explorer and Amazon CloudWatch, to gather real-time cost and utilization metrics. Empirical measurements during the evaluation phase revealed that the API latency remained below 400 milliseconds per call, while the average end-to-end system response time (from initial Lex query to final Lex response) was approximately 2.1 seconds, thus meeting the design goal of sub-3-second interaction latency.

Data persistence and audibility were improved through the use of Amazon DynamoDB, where session-specific attributes allowed efficient lookup of historical query logs and metadata. Furthermore, long-term storage of aggregated metrics in Amazon S3 facilitated the support of advanced analytical workloads, including periodic model retraining using SageMaker pipelines. This design ensures both operational responsiveness and extensibility for future enhancements.

B. Cost Efficiency and Resource Utilization

The serverless architecture leveraged the pay-per-use pricing model to achieve low daily costs. Step Functions streamlined workflow execution more effectively than long-running Lambdas, contributing to compute efficiency. Storage optimizations, including Intelligent-Tiering and lifecycle transitions to Glacier, led to a 22 percent reduction in S3 costs over one month. These results demonstrate tangible benefits in both compute and storage dimensions.

C. Machine Learning Model Performance

The Random Forest classifier was evaluated on a held-out test dataset of 1,100 samples (800 optimized resources and 300 underutilized resources). While overall accuracy was 60 percent, class imbalance affected predictive performance. The model performed well on Class 0 (optimized resources) with an F1-score of 0.67 but struggled on Class 1 (underutilized resources), achieving an F1-score of 0.29. The macro-average F1-score of 0.48 indicates opportunities for improving model performance, particularly in handling minority classes.

D. Practical Impact

The solution retrieves and presents AWS service usage data—including cost and utilization metrics—over specified time ranges and provides actionable recommendations such as downsizing EC2 instances. The conversational interface

Metric	Class 0 (Optimized)	Class 1 (Underutilized)	Macro Avg	Weighted Av
Precision	0.71	0.25	0.48	0.59
Recall	0.63	0.33	0.48	0.60
F1-score	0.67	0.29	0.48	0.59
Accuracy	–	–	0.60	0.60

Fig. 4. Accuracy Metrics.

powered by Amazon Lex allows users to query service usage and receive optimization insights naturally. The system enables proactive identification of resource inefficiencies and supports cost control actions, demonstrating the feasibility and value of combining serverless services with AI-driven analytics for cloud cost management. Despite these limitations, the model effectively identified optimized resources with reasonable precision, supporting the goal of proactive resource management.

E.Project Outcome

Retrieve and present service usage data for a specified time range, including cost and utilization metrics. Analyze resource utilization (in this case, potentially for an "SQS" service and EC2 instances). Provide actionable recommendations for cost optimization, such as suggesting downsizing EC2 instances to a more suitable type (e.g., t2.xlarge). Offer a conversational interface that allows users to easily query and receive information and recommendations.

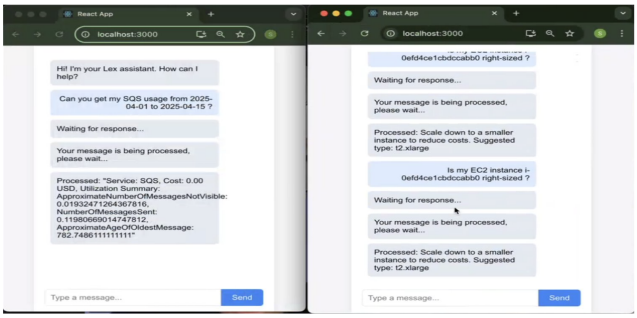

Fig. 5. Final Result

F. Tool Cost and Usage

Our proposed tool leverages a serverless architecture, inherently minimizing operational overhead and resulting in significant cost optimization by eliminating the need for continuous resource provisioning and management. The presented solution adopts a fully serverless paradigm, ensuring cost efficiency through its pay-per-use model and virtually eliminating operational overhead associated with traditional server management. By employing a serverless design, our tool achieves substantial cost optimization and avoids the typical overhead of maintaining dedicated infrastructure, leading to a highly efficient and resource-conscious solution. The architecture of our tool is entirely serverless, which directly translates to a cost-optimized solution with little to no operational overhead, as resources are dynamically allocated and managed by the cloud provider. We introduce a serverless and cost-optimized

tool characterized by its minimal operational overhead. This is achieved through the inherent scalability and pay-as-you-go nature of serverless computing.

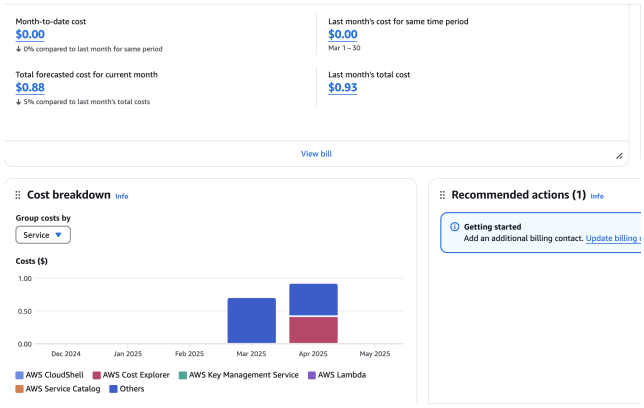

Fig. 6. Tool Costing and usage.

V. Discussion, Limitations, and Next Steps

A. Discussion

Our Serverless AI Chatbot presents a novel approach to cloud cost optimization by leveraging natural language processing to interpret AWS usage queries and provide actionable recommendations. The event-driven architecture using Amazon Lex, Lambda, and SQS supports scalable and asynchronous processing effectively.

Initial testing yielded approximately 60% accuracy in optimization recommendations, primarily limited by the size and quality of the training dataset rather than architectural flaws. Despite this, the system demonstrates strong potential for translating raw AWS cost data into meaningful insights through natural language.

The integration of AWS Step Functions, Cost Explorer, and CloudWatch enables effective workflow coordination and resource visibility. While challenges were encountered during development, the project lays a solid foundation for further enhancements in intelligent cost management.

B. Limitations

Our implementation faced several significant limitations:

- **No actual production data for testing:** Reliance on dummy data for training our machine learning models significantly impacted prediction accuracy, limiting our ability to test the system under realistic conditions.
- **Resource constraints:** AWS account limitations restricted our ability to deploy resources at the scale needed for comprehensive testing, particularly when working with SageMaker model training.
- **Spot instance availability challenges:** Training machine learning models required substantial computational resources, but the availability of cost-effective spot instances was inconsistent, affecting our development timeline.
- **Limited Amazon Lex intents:** We implemented only two primary intents (`CheckAWSUsage` and `CheckInstanceSize`), which restricts the range of queries the chatbot can effectively respond to.
- **AWS service throttling limits:** Standard AWS throttling limits on Lambda invocations, Cost Explorer API calls, and CloudWatch metric retrievals created bottlenecks when processing multiple concurrent requests.

C. Next Steps

Based on our findings and identified limitations, we plan the following improvements:

- **Production environment testing:** Deploy the solution in an actual production environment with real AWS usage data to validate its effectiveness and refine recommendation algorithms.
- **Machine learning accuracy improvement:** Enhance model accuracy from the current 60% to above 90% by incorporating real usage patterns, implementing feature engineering, and exploring more advanced model architectures.
- **Response time optimization:** Reduce the turnaround time for processing queries by optimizing Lambda functions and implementing caching strategies for frequently accessed data.
- **Intent expansion in Amazon Lex:** Add additional intents to handle a wider range of cost optimization queries, including budget forecasting, anomaly detection, and service-specific optimization recommendations.
- **Service coverage expansion:** Extend beyond EC2, Lambda, and S3 to include optimization for additional AWS services like RDS, ECS, and DynamoDB.
- **Generative AI integration:** Incorporate generative AI capabilities to provide more contextual and detailed explanations of cost optimization recommendations rather than fixed response templates.
- **Reinforcement learning implementation:** Apply reinforcement learning techniques to allow the model to improve based on the outcomes of implemented recommendations, creating a feedback loop that continuously enhances recommendation quality.

References

[1] Amazon Web Services, "Amazon Lex: Build conversational interfaces for any application," AWS, [Online]. Available: https://aws.amazon.com/lex/. [Accessed: 29-Mar-2025].

[2] Amazon Web Services, "Amazon SageMaker: Build, train, and deploy machine learning models," AWS, [Online]. Available: https://aws.amazon.com/sagemaker/. [Accessed: 29-Mar-2025].

[3] Amazon Web Services, "AWS Lambda: Run code without thinking about servers," AWS, [Online]. Available: https://aws.amazon.com/lambda/. [Accessed: 29-Mar-2025].

[4] M. Richardson and M. J. J. P. V. Jansen, "Serverless Computing: Economic and Architectural Impact," in *Proceedings of the 2018 IEEE International Conference on Cloud Computing*, San Francisco, CA, USA, 2018, pp. 322-329. doi: 10.1109/CloudCom.2018.00057.

[5] R. S. O. McKinney, "Machine Learning for Cloud Cost Optimization: A Predictive Approach," *Journal of Cloud Computing*, vol. 6, no. 2, pp. 87-96, Apr. 2023.