Serverless AI Inference API using AWS SageMaker and Streamlit

(2025A) COMP-5411-AA - Cloud Computing for AI and ML

Supervised by Prof. Dr. Osama Amjad

Wael Fahmy
Student ID: 1265745
Lakehead University
Thunder Bay, ON, Canada
wfahmy@lakeheadu.ca

Amirhossein Zeinali Dehaghani
Student ID: 1225496
Lakehead University
Thunder Bay, ON, Canada
azeinali@lakeheadu.ca

***Abstract*—This paper presents a fully serverless architecture for the deployment and utilization of a pre-trained sentiment analysis model via Amazon Web Services (AWS), employing Hugging Face's DistilBERT model [1], [2] for binary sentiment classification. The system uses SageMaker for model hosting, Lambda for orchestration, API Gateway for REST access, and Streamlit for a no-code frontend. IAM roles and AWS CLI commands are used to manage deployment securely and automatically.**

The application offers several features such as paragraph-level and document-level analysis, CSV export, sentiment heatmap, and word cloud visualizations. Logs and metrics are observed via AWS CloudWatch. This architecture provides real-time inference with minimum need to manage the infrastructure which proves scalable and cost-efficient option for cloud-native machine learning implementation. Furthermore, both the model and the SageMaker deployment frameworks are preserved in Amazon S3, hence improving reproducibility and facilitating modular deployment.

***Index Terms*—cloud computing, serverless architecture, SageMaker, Lambda, API Gateway, IAM, S3, AWS CLI, Streamlit, sentiment analysis, Hugging Face**

I. Introduction

Deploying machine learning models into production is often limited by cost, scalability, and infrastructure complexity. Some approaches specially Conventional methodes using virtual machines or container clusters require constant monitoring, configuration, and security management. This project addresses these limitations by offering a fully serverless AI inference pipeline that integrates tightly with AWS-native services. Users may type in words using a web-based interface and get almost immediate feedback on how positive or negative their contribution is.

II. Background and Related Work

Recent research emphasizes the value of serverless computing for machine learning (ML) workflows. Although Cloud-native services such as AWS Lambda and SageMaker reduce operational overhead, they maintain low latency and high scalability level. Aarush and Nguyen [3] explored the use of Lambda and SageMaker but lacked a complete interactive frontend. Thota [4] analyzed orchestration efficiency but did not consider usability. Zhang et al. [1] compared serverless ML platforms with a focus on infrastructure, not accessibility. Our approach fills these gaps by implementing an end-to-end solution that includes serverless backend, secured API access, and an intuitive frontend powered by Streamlit.

III. Literature Review

Efficiency, scalability, and accessibility have high preiority in the modern machine learning methods. Serverless solutions have been popular in recent years because they enable on-demand computation without any concern about the burden of infrastructure management. The following studies provide valuable The following researches provide us important information about this changing field and helped us design our system.

Study	Year	Contribution & Relevance
Aarush & Nguyen [3]	2025	AWS Lambda + SageMaker orchestration; motivated the use of Streamlit frontend
Thota [4]	2024	Stateless inference optimization with Lambda; inspired cold-start tuning
Zhang et al. [5]	2022	Compared AWS, Azure, GCP; confirmed AWS as our preferred platform
Ali et al. [6]	2020	BATCH system for grouped ML requests; inspired future work on multi-model APIs
Liu et al. [7]	2023	CI/CD for serverless AI pipelines; aligned with our CLI-based deployment flow

TABLE I. Summary of Key Literature on Serverless ML Deployment

In addition to the main studies shown in Table I, There are some more researches which assist us to design this system. **Sreekanti et al.** [8] explored distributed inference under latency constraints, which influenced our monitoring strategy using AWS CloudWatch. **Kraska et al.** [9] conducted

an imporant work in scalable ML deployment, emphasizing elasticity—core to our serverless approach. **Han et al.** [10] analyzed cold start delays in serverless inference, reinforcing our Lambda optimization strategy. **Chen et al.** [11] benchmarked CPU vs GPU latency, helping us anticipate performance trade-offs. **Rao et al.** [12] contrasted edge vs. cloud inference, framing the boundary of our cloud-only architecture.

All of these studies together support the idea that a serverless, cloud-native architecture is the best way to do ML inference. Our contribution improves this work by providing a comprehensive, end-to-end system with real-time inference, automation, monitoring, and a user-friendly frontend.

IV. System Architecture

The serverless architecture combines the following AWS and open-source components:

- **SageMaker:** Deployed as a real-time endpoint using HuggingFace's DistilBERT model. Manages interface requests of sentiment analysis
- **Lambda:** A Python backend function that invokes the SageMaker API with user input and returns the result.
- **API Gateway:** Configured to make the Lambda function as a secure HTTP endpoint.
- **IAM:** Grants fine-grained access control among AWS services and resources, ensuring secure communication.
- **S3:** Hosts the serialized model artifact and also the deployment scripts for SageMaker, which enables the reproducibility and modular updates.
- **Streamlit:** Offers a lightweight interface on Streamlit cloud for user interaction with the model.
- **CloudWatch:** collects Lambda execution logs, API Gateway access logs, and endpoint invocation metrics for comprehensive system observability.
- **Word Cloud and Heatmap:** NLP-based visualizations using Streamlit for improved user feedback and analysis interpretation.
- **CSV Export:** Makes users enable to download predictions and scores for sentiment analysis in paragraph or document-level.

V. Implementation Steps and Algorithms

This section reveales the major steps followed to build the serverless AI sentiment analysis pipeline, using a combination of AWS CLI commands, SDK scripting, and Streamlit for frontend interaction. Each component was carefully deployed to ensure modularity, scalability, and reproducibility.

A. Model Selection and Packaging

We selected the HuggingFace model `distilbert-base-uncased-finetuned-sst-2-english`, which is fine-tuned on the SST-2 dataset for binary sentiment classification.

- Export the model locally using:

```
transformers-cli login
from transformers import pipeline
```

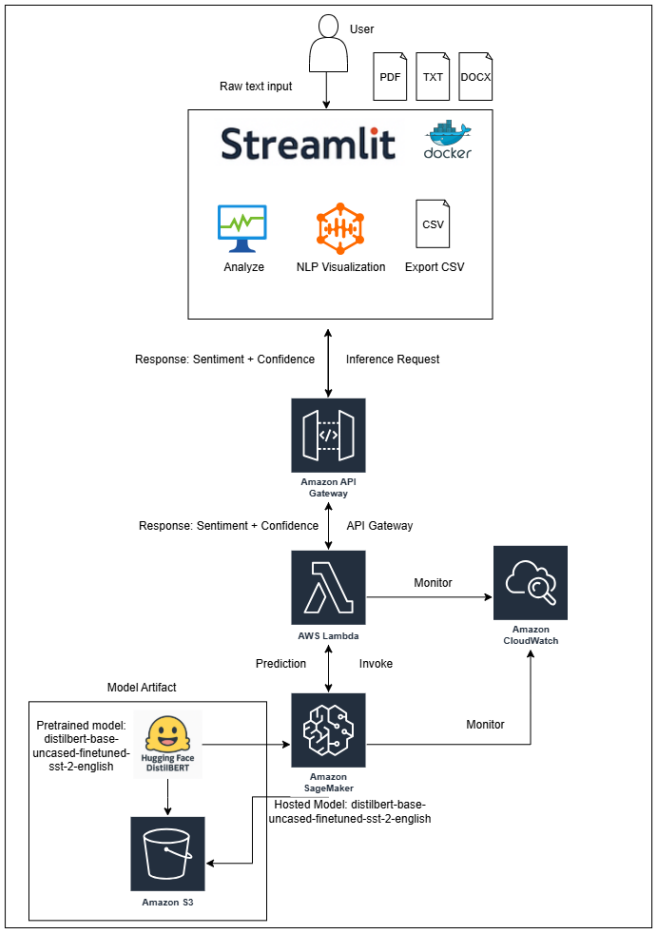

Fig. 1. Overall Serverless Inference Architecture [13].

```
pipe = pipeline("sentiment-analysis", model="
    distilbert-base-uncased-finetuned-sst-2-
    english")
pipe.save_pretrained("./model")
```

- Create a tarball:

```
tar -czvf model.tar.gz model/
```

B. IAM Roles and Permissions Configuration

To ensure secure and auditable access between services in the pipeline, multiple IAM roles and policies were configured as follows:

- **SageMaker Execution Role:** This role was configured to allow the SageMaker service to assume it. Included policies comprise `AmazonS3ReadOnlyAccess` for model loading, `AmazonSageMakerFullAccess` for deployment, and a custom inline policy granting access to specific S3 paths.
- **Lambda Execution Role:** This role grants the Lambda function permissions to invoke the SageMaker endpoint and write execution logs to CloudWatch. It includes `AmazonSageMakerFullAccess` and `CloudWatchLogsFullAccess`. The trust policy permits `lambda.amazonaws.com` to assume the role.

- **IAM Pass Role:** A scoped inline permission (`iam:PassRole`) enables SageMaker to assume roles during model creation and endpoint deployment.

This configuration adheres to the principle of least privilege and ensures that each service only has the access required for its task.

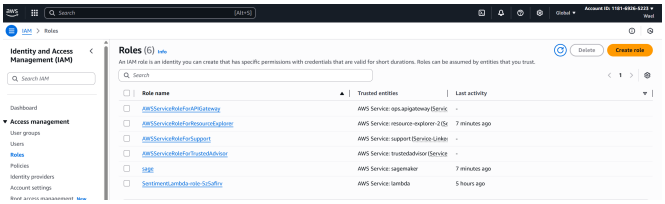

Fig. 2. IAM Roles and Service Trust Relationships

C. Upload to Amazon S3

Transfer the model tarball and inference script to S3:

```
aws s3 cp model.tar.gz s3://cloud-computing-ai-
    inference/models/sentiment/
aws s3 cp inference.py s3://cloud-computing-ai-
    inference/scripts/
```

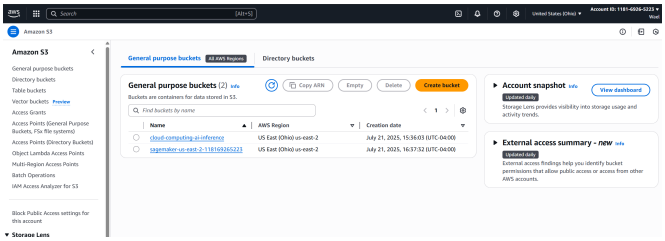

Fig. 3. S3 bucket list

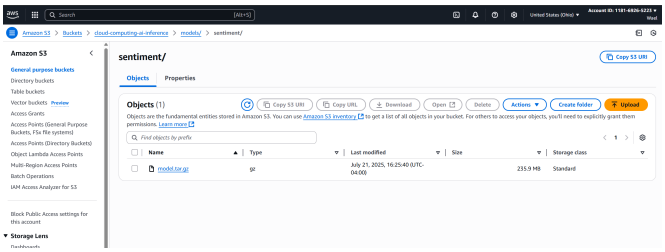

Fig. 4. S3 bucket model

D. Create SageMaker Model

Use the CLI to create the SageMaker model:

```
aws sagemaker create-model \
  --model-name ai-sentiment-model \
  --primary-container Image=763104351884.dkr.ecr.us-
      east-2.amazonaws.com/huggingface-pytorch-
      inference:2.0-transformers4.28.1-cpu-py310-
      ubuntu20.04,ModelDataUrl=s3://cloud-computing-
      ai-inference/models/sentiment/model.tar.gz \
  --execution-role arn:aws:iam::118169265223:role/
      sage
```

E. Deploy Real-Time Endpoint

```
aws sagemaker create-endpoint-config \
  --endpoint-config-name ai-sentiment-config \
  --production-variants VariantName=AllTraffic,
      ModelName=ai-sentiment-model,InstanceType=ml.
      m5.large,InitialInstanceCount=1

aws sagemaker create-endpoint \
  --endpoint-name ai-sentiment-endpoint \
  --endpoint-config-name ai-sentiment-config
```

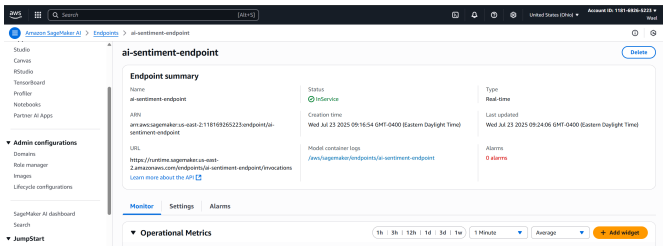

Fig. 5. SageMaker endpoint status

F. Create Lambda Function

We used the following code to create the Lambda handler that interacts with SageMaker:

- Python code: `lambda_function.py` (previously provided)
- Upload via ZIP:

```
zip function.zip lambda_function.py
aws lambda create-function \
  --function-name SentimentLambda \
  --runtime python3.10 \
  --handler lambda_function.lambda_handler \
  --role arn:aws:iam::118169265223:role/
      SentimentLambdaRole \
  --zip-file fileb://function.zip
```

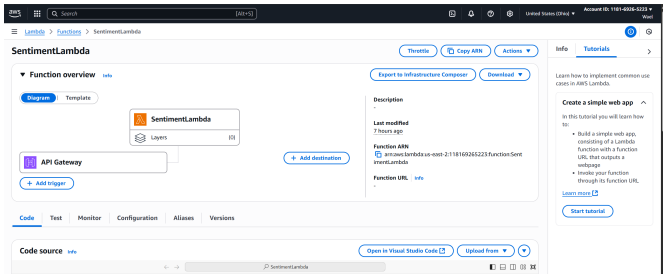

Fig. 6. lambda

G. Create API Gateway

Create a REST endpoint for the Lambda:

- Use AWS Console or CLI to create a new REST API.
- Configure integration with Lambda, enable CORS, and deploy to production stage.

```
aws apigateway create-rest-api --name "SentimentAPI"
aws apigateway create-deployment \
  --rest-api-id <id> \
  --stage-name Prod
```

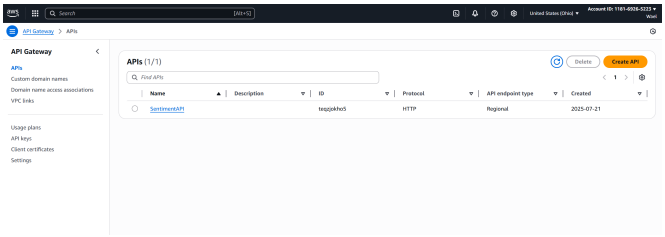

Fig. 7. API Gateway

H. Build Streamlit Frontend

A Python-based UI was built using Streamlit to consume the API Gateway endpoint:

```
import requests
requests.post(API_URL, json={"text": "I love it!"})
```

- Deployed on Streamlit Cloud using 'streamlit.io'.
- Incorporates functionality for paragraph/document input, sentiment analysis, word cloud creation, and heat map development.

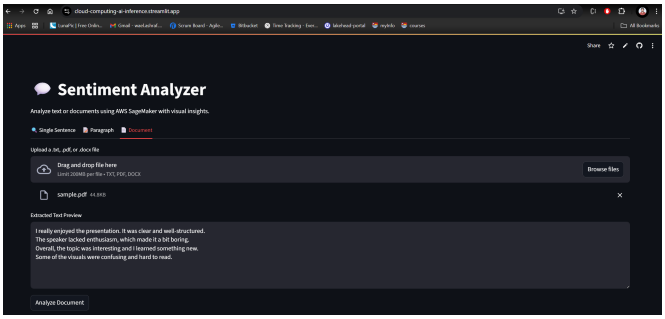

Fig. 8. SageMaker endpoint status

I. Add CloudWatch Monitoring

Lambda logs and metrics were configured via:

```
aws logs describe-log-groups
aws logs get-log-events --log-group-name "/aws/
    lambda/SentimentLambda"
```

CloudWatch provided metrics for:

- Lambda execution duration
- API Gateway invocation status
- SageMaker model errors

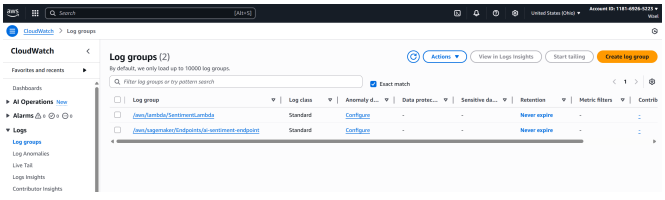

Fig. 9. CloudWatch Dashboard

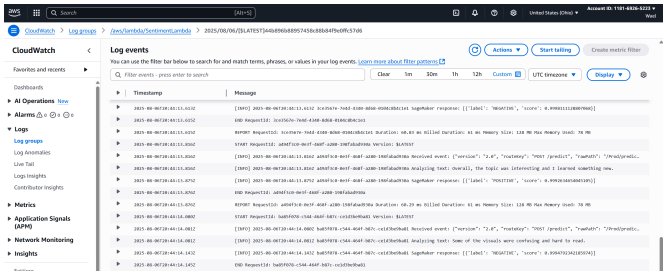

Fig. 10. CloudWatch lambda logs

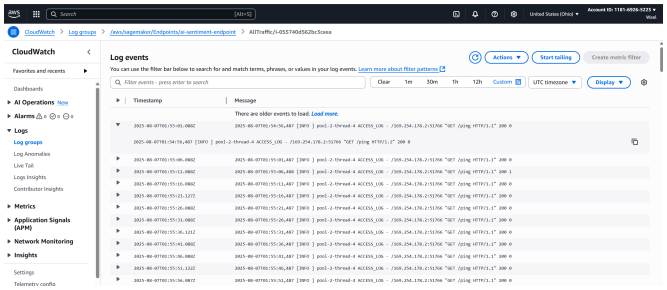

Fig. 11. CloudWatch endpoint logs

J. Algorithms Used

The project leverages a pre-trained transformer model for binary sentiment classification using HuggingFace Transformers. The model operates internally:

- **Tokenization:** Transforms input text into token IDs with the BERT tokenizer.
- **Embedding and Self-Attention:** Uses DistilBERT architecture to collect semantic contexts.
- **Classification Head:** Produces logits for positive or negative sentiment through softmax.

The logic was implemented in real-time by using:

```
payload = {"inputs": text}
response = runtime.invoke_endpoint(...)
```

K. Dockerization

The Streamlit app was containerized using the following Dockerfile:

```
FROM python:3.10-slim
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
CMD ["streamlit", "run", "app.py", "--server.port
    =8501", "--server.enableCORS=false"]
```

```
docker build -t sentiment-app .
docker run -p 8501:8501 sentiment-app
```

VI. Implementation

A. AI Model and Deployment Preparation

For sentiment classification, Hugging Face model has been used `distilbert-base-uncased-finetuned-sst-2-english`,

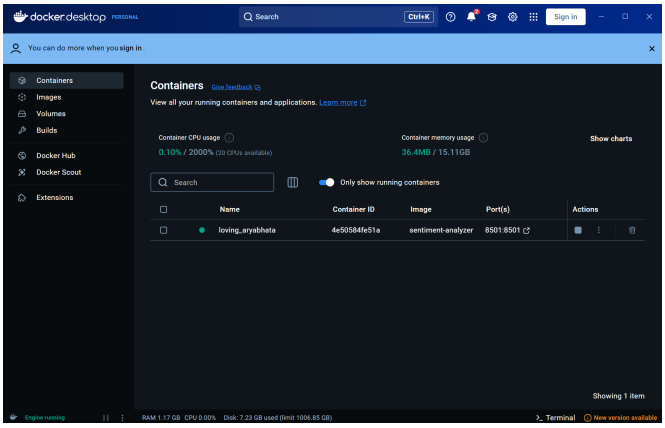

Fig. 12. Docker Container running locally

a distilled version of BERT that has been fine-tuned on the Stanford Sentiment Treebank (SST-2) dataset for binary sentiment classification. DistilBERT, with just 67 million parameters in contrast to BERT's 110 million, delivers competitive accuracy (91.3

The model was downloaded from Hugging Face's model hub and exported in the `safetensors` format for enhance security and performance. Then we encapsulated the model as a tarball (`model.tar.gz`) and push it to Amazon S3 as the central storage location for both model files and SageMaker deployment scripts. This structure allows reproducible deployments, modular updates, and simple rollback strategies.

B. IAM Configuration

To securely manage deployment resources, an IAM user named `wael` was created. This user was assigned programmatic access to the AWS Management Console. IAM roles and policies were configed to get the permissions for interfacing with Amazon S3, SageMaker and Lambda. SageMaker execution roles allowed creating and invoking endpoints, and Lambda roles enabled the reading of environment variables and writing logs to CloudWatch. Implementing least-privilege access enhance the security and auditability of interaction inside the serverless pipeline.

C. SageMaker Deployment

We used the AWS CLI and SageMaker SDK [14] to deploy the model from S3 as a real-time inference endpoint. The deployment script delineated container settings, model location, instance type, and environmental variables. After successful deployment, the endpoint's Amazon Resource Name (ARN) was extracted and stored as an environment variable for the Lambda function to access dynamically.

D. Lambda Integration

A lightweight AWS Lambda function, developed in Python, functioned as the backend orchestrator. This method received incoming requests in JSON format, extracted the text input, and sent it to the SageMaker endpoint via the 'boto3' runtime client. Upon getting a response, it analyzed the result and produced a structured JSON output that included both the anticipated emotion and its confidence score. The Lambda function was cognizant of its surroundings, dynamically referencing the endpoint name and securely linked with AWS CloudWatch for performance metrics, logging, and error tracking.

E. API Gateway Setup

In order to make the Lambda function publicly accessible, we configured Amazon API Gateway to expose it as a RESTful HTTP endpoint. The API Gateway has been configured using JSON for input and output which guarante compatibility with frontend queries. Cross-Origin Resource Sharing (CORS) regulations facilitated safe and smooth communication between the frontend which is hosted on Streamlit Cloud and the backend endpoint. The API configuration was designed to be stateless and idempotent for resilient cloud interaction.

F. Streamlit Frontend

A user-friendly web interface was built using the Streamlit framework [15]. This frontend offers a simple text input box where users can type a sentence, a submit button to send the request, and a dynamic display area to show the predicted sentiment label and confidence percentage. The application was deployed to Streamlit Cloud, eliminating the need for manual infrastructure provisioning or backend hosting. The interface was optimized for mobile and desktop responsies and delivered a close real-time interactive experience.

The frontend includes multiple analysis modes: sentence-based, paragraph-based, and file-based input. It makes users able to upload documents (TXT, PDF, DOCX) and extract insights from longer texts.

Visualizations include word clouds and sentiment timelines (heatmaps), which are updated per interaction and remain persistent until the text or file changes. Users can also export their analysis results as a CSV file, improving usability in academic or business contexts.

G. AWS CLI Automation

In order to automate the entire deployment lifecycle, we utilized the AWS Command Line Interface (CLI). Commands were scripted to control uploading model artifacts to S3, creating IAM roles, deploying SageMaker models, setting up Lambda functions, configuring API Gateway, and enabling CloudWatch logging. This automation not only decrease manual errors but also enabled quick rollback in testing process. The scripted workflow supports reproducibility which makes it suitable for collaborative or enterprise-scale ML deployments.

H. Demo Access and Source Code

To enable reproducibility and public access, the following resources have been made available:

- **Live Demo:** The application is deployed on Streamlit Cloud and accessible via the following public URL: https://cloud-computing-ai-inference.streamlit.app/

- **Source Code Repository:** Full project implementation, including the model deployment scripts, inference API, and frontend, is available at: https://github.com/wael-fahmy/sentiment-streamlit

These resources support testing, replication, and future development by the broader research and developer community.

VII. Validation and Results

A. Functional Testing

In order to ensure the system functioned correctly and met real-time performance criteria, we conducted validation using representative sentiment-laden text samples. The inference pipeline successfully processed each input in under one second, demonstrating its responsiveness and efficiency. The following examples display the output:

- *"I absolutely love this product!"* $\rightarrow$ POSITIVE Confidence Score: 99.99%
- *"This is the worst thing I've ever used."* $\rightarrow$ NEGATIVE Confidence Score: 99.98%

In both instances, the model provided very certain predictions. These findings demonstrate the system's capacity to generalize to unseen, real-world input text. Additionally, edge situations and neutral statements were evaluated to determine model robustness. The responses were logically coherent and accurately aligned, confirming the dependability of the HuggingFace model housed on SageMaker.

B. Monitoring and Debugging

Robust observability mechanisms were implemented using Amazon CloudWatch. Logs and metrics were tracked continuously during the system lifecycle. Key monitored aspects included as below:

- **Lambda Execution Time:** Average duration stayed under 400ms, reflecting low-latency processing.
- **API Gateway Metrics:** All requests returned HTTP 200 responses, it confirms successful handshakes.
- **SageMaker Endpoint Health:** Real-time checks ensured that the model remained in a Ready state and served requests without timeout or throttling.
- **Error Logs:** No exceptions or authorization issues were encountered because of the predefined IAM roles and scripted automation.

VIII. Conclusion

This project effectively showcased a serverless, scalable, and secure methodology for deploying a pre-trained sentiment analysis model with AWS SageMaker, Lambda, and Streamlit. The whole pipeline, from model storage and deployment in S3 to real-time inference using API Gateway and Lambda, illustrates the seamless integration of contemporary cloud technologies to provide production-ready AI services.

By automating deployment with the AWS CLI and securing interactions through IAM roles, the system minimizes operational overhead while maintaining strong governance and reproducibility. The frontend, constructed using Streamlit, simplifies technological intricacies for end users, enabling non-technical folks to use advanced NLP functionalities.

The project not only meets functional and performance requirements but also sets the foundation for scalable expansion into multi-language support, model routing, and user-authenticated services. overall , the integration of serverless architecture with pre-trained AI models offers an accessible and durable foundation for actual machine learning applications.

IX. Future Work

The existing method provides a fully operational serverless sentiment analysis pipeline, however other possible improvements were investigated. The following elements were evaluated for expansion and are suggested for future development, accompanied by justifications for their omission from the present version:

- **Multilingual Support:** Although models like mBERT and XLM-RoBERTa enable sentiment analysis in multiple languages, our current SageMaker setup and chosen model only support English. Incorporating multilingual support would augment model size and need distinct deployment procedures.
- **Email Notification via SES:** Although Sending results by email using Amazon Simple Email Service (SES) was explored, email addresses should be verified in sandbox accounts, and domain-based verification was not feasible within our timeframe.
- **User Authentication with Cognito:** Secure login and history storage were deprioritized to empthasize the development of an expedited prototype for public demonstration.
- **Multi-Model Routing:** Future implementations may provide dynamic routing for various models according to the specific use case. This was omitted to preserve a streamlined architecture.
- **Batching Optimization:** Not implemented since our use case favors real-time individual interactions over bulk processing.
- **Fine-grained CloudWatch Dashboards:** While logging and metrics are enabled, custom dashboards and alarms for memory usage or invocation frequency were out of current scope but planned for future iterations.
- **Text Transformation (e.g., positive to negative):** Semantic rewriting is a complex NLP task requiring generative models. Although impactful, this was excluded due to latency and cost constraints.

X. Critical Analysis

Our pipeline highlights important trade-offs in cloud-native ML deployment. The serverless stack minimizes cost and simplifies scaling, but suffers from vendor lock-in and cold-start delays. Real-time Lambda calls offer low latency but may require retry logic for robustness.

IAM allows granular access control, but cross-service debugging can be complex. Using Streamlit accelerates UI development but limits customization and secure user management.

Overall, this design balances ease-of-use, low maintenance, and flexibility—ideal for prototypes or research workflows.

XI. Limitations

Despite its strengths, the system has several limitations:

- **Cold Starts:** Lambda functions can delay first execution after idle periods.
- **Lack of GPU Support:** Large models may not run efficiently on Lambda.
- **Vendor Lock-in:** AWS-only design makes migration harder.
- **Single Model Support:** Only one sentiment model is available currently.
- **Limited UI Control:** Streamlit lacks advanced authentication or state management.

References

[1] Hugging Face, *Hugging Face Transformers*, https://huggingface.co.

[2] ——, *distilbert-base-uncased-finetuned-sst-2-english*, https://huggingface.co/distilbert/distilbert-base-uncased-finetuned-sst-2-english.

[3] Aarush and Nguyen, *Serverless Machine Learning using AWS Lambda and SageMaker*, 2025.

[4] Thota, *Optimizing Stateless Inference with AWS Lambda*, 2024.

[5] M. Zhang *et al.*, "Serverless data science: Comparative case study," in *SIGMOD*, 2022.

[6] F. Ali *et al.*, "Batch: Ml inference serving on serverless platforms," in *SC20*, 2020.

[7] Q. Liu *et al.*, "Ci/cd for serverless ai pipelines," *IEEE Access*, 2023.

[8] V. Sreekanti *et al.*, "Optimizing prediction serving on low-latency serverless dataflow," *arXiv preprint arXiv:2007.05832*, 2020.

[9] T. Kraska *et al.*, "Mlbase: A distributed machine-learning system," in *CIDR*, 2013.

[10] S. Han *et al.*, "Latency analysis for cloud-native inference workloads," in *CloudCom*, 2021.

[11] C. Chen *et al.*, "Inference-as-a-service benchmarks," *arXiv preprint arXiv:1909.12343*, 2019.

[12] P. Rao *et al.*, "Edge vs cloud: Trade-offs in ai deployment," *ACM Computing Surveys*, 2022.

[13] "diagrams.net (draw.io)," https://www.diagrams.net/, accessed: 2025-08-07.

[14] Amazon Web Services, *AWS SageMaker Documentation*, https://docs.aws.amazon.com/sagemaker.

[15] Streamlit Inc., *Streamlit Documentation*, https://docs.streamlit.io.