National College of Ireland

Real-Time Cloud-Based Anomaly Detection

MSc Research Project

Cloud Computing

Gokul Girish Kumar

Student ID: 23297379

School of Computing

National College of Ireland

Supervisor: Punit Gupta

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Gokul Girish Kumar		
Student ID:	23297379		
Programme:	MSc Cloud Computing	**Year:**	2024
Module:	MSc Research Project		
Supervisor:	Punit Gupta		
Submission Due Date:	15/09/2025		
Project Title:	Real-Time Cloud Based Anomaly Detection		
Word Count:	7684	**Page Count:**	19

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Gokul Girish Kumar

Date: 15/09/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Real-Time Cloud-Based Anomaly Detection

Gokul Girish Kumar

23297379

Abstract

The rapid increase in cloud computing has made robust, scalable, and intelligent security solutions more critical than ever. Traditional security systems often struggle to keep pace with the dynamic nature of cloud environments and the increasing sophistication of cyber threats. This research presents the end-to-end design, implementation, and deployment of a scalable, real-time intrusion and anomaly detection system tailored for cloud security. The project's unique contribution lies in its novel integration of a deep learning model with a fully serverless cloud architecture. A Long Short-Term Memory (LSTM) neural network is trained on the comprehensive CICIDS2017 network intrusion dataset to accurately distinguish between benign and malicious network traffic. The trained model is brought to production through a highly innovative and entirely serverless pipeline on Amazon Web Services (AWS), with SageMaker Serverless Inference used as hosting, the AWS Lambda to squeeze requests, and the Amazon API Gateway to publish the functionality as a publicly-accessible endpoint. It would equip the researcher with a detailed technical blueprint, proving a competent, economical, and highly scalable solution, to the new cloud-based cybersecurity headaches.

Keywords: Anomaly Detection, Intrusion Detection, Deep Learning, LSTM, Serverless Architecture, AWS SageMaker, Cloud Security

1. Introduction

1.1 Background

A global migration to the cloud of organizations and their essential and valuable organizational information is taking place. The nature of cybersecurity has completely changed. Cloud settings have flexibility and scalability never seen before and even introduce new attack vectors as well as complicated security environments challenges. Conventional security defense (like rule-based firewalls and signature-based) are already becoming obsolete, becoming inadequate intrusion detection systems. Such legacy systems tend to be reactive; they detect threats once a known signature is found and therefore, they are left vulnerable. How they are exposed to new, zero-day attacks. Moreover, they have poor scaling efficiency, massive data scale and data flow that is created in contemporary, dispersed cloud applications.

To address these shortcomings, cloud security is shifting into the direction of a smart, adaptive defense mechanisms. Machine learning, and in particular deep learning, has become a strong trend in developing the next generation of security systems. Unlike the traditional ways, deep learning models have the ability to autonomously identify complex patterns and minute aberrations if it is applied to identify baseline behavior in large databases. This enables them to identify anything that they had not seen. High precision guess anomalies and intrusions. The project will concentrate on tapping into this potential so that a proactive and intelligent security solution is developed. The gist of the whole thing is to transcend the simple threat detection to design a system that is designed-in-cloud-first, which is a system that is not just smart, but also scalable, resilient and operationally efficient in a real-time setting.

1.2 Aim of the Study

The main purpose of the research is the creation and implementation of a scalable and real time intrusion and anomaly detection system targeted at cloud security. The project aims to give a complete step-by-step technical guide that will cover the lifecycle of the data ingestion and data preparation to the model training and its ultimate deployment as a publicly facing API.

The distinct advantage of the research is that it is a combination of a complex deep learning framework and a fully serverless architecture pattern on the AWS platform. This strategy solves two important security issues in the contemporary world:

1. **Detection Accuracy:** By employing a Long Short-Term Memory (LSTM) network, a type of recurrent neural network, the system can effectively model sequential data like network traffic flows to identify complex anomalies.

2. **Scalability and Cost-Effectiveness:** By leveraging a serverless architecture—specifically AWS SageMaker Serverless Inference, AWS Lambda, and Amazon API Gateway—the system eliminates the need for managing underlying infrastructure, automatically scales with demand, and follows a pay-per-use model, ensuring operational efficiency.

The study utilizes the well-established CICIDS2017 network intrusion dataset for training and evaluation, ensuring the model is grounded in realistic and diverse network traffic scenarios.

1.3 Research Questions

This study is guided by the following core research questions:

1. How can a deep learning model, specifically a 1D Long Short-Term Memory (LSTM) network, be effectively trained and evaluated using the CICIDS2017 dataset to accurately classify network traffic as either benign or anomalous?

2. What is the end-to-end process for deploying a trained TensorFlow/Keras model into a fully serverless, scalable, and real-time inference architecture using a combination of AWS SageMaker Serverless, AWS Lambda, and Amazon API Gateway?

3. How effective is this integrated serverless architecture in providing a practical, high-performance, and publicly accessible API for real-time anomaly detection?

The rest of this report is divided as follows: Chapter 2 provides researchers with an overview of the related literature, including current trends, limitations, and research gaps in intrusion detection. In Chapter 3, the methodology of conducting research and workflow used in this research is described. Chapter 4 explains the design requirements of the proposed system and Chapter 5 explains the implementation of the system. Within Chapter 6, the performance of the system under scrutiny is explored and analysed based on the metric and comparison benchmarks. Finally, the work is completed with Chapter 7 that points out the most important findings, limitations and perspectives of further research.

2. Related Work

The chapter provides a critical review of the literature on the topic of network intrusion detection, with particular interest on the transition from developing conventional signature-based methodologies to the latest deep learning and serverless deployment implementation. It contains strengths, weaknesses, and research gaps upon which a proposed solution is based on.

The field of cybersecurity, particularly Network Intrusion Detection Systems (NIDS), has undergone a significant transformation. Driven by the dual pressures of increasingly sophisticated cyber threats and the mass migration of infrastructure to the cloud, the need for intelligent, scalable, and automated security solutions has never been more critical. This chapter provides a critical review of the key literature that charts the evolution of intrusion detection methodologies, from early signature-based systems to the advanced deep learning models that define modern research.

The review first examines the foundational role of traditional and classical machine learning approaches, acknowledging their contributions while highlighting their inherent limitations in the face of new, zero-day attacks. It then progresses to the core of contemporary research: the application of deep learning. A significant focus is placed on Recurrent Neural Networks (RNNs) and their more advanced variants, Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks, which are exceptionally well-suited to analyzing the sequential nature of network traffic. The chapter also investigates the use of these models in conjunction with the CICIDS2017 dataset, a modern benchmark that has become central to NIDS research.

Finally, the review shifts from model theory to practical implementation, contrasting traditional server-based deployment with the rise of serverless computing for Machine Learning Operations (MLOps). By synthesizing these distinct but interconnected areas of research, this chapter identifies a critical research gap at the intersection of advanced deep learning models and fully serverless, real-time deployment architectures, thereby establishing the academic and practical context for this project.

2.1 Evolution of Intrusion Detection Methodologies

The progression of NIDS can be understood as a direct response to the increasing complexity of cyber threats. Early systems were predominantly **signature-based**, operating like antivirus software by matching network traffic against a database of known malicious patterns (Buczak and Guven, 2016). While highly effective at detecting known threats with a low false-positive rate, their reactive nature renders them incapable of identifying novel or "zero-day" attacks (Khraisat et al., 2019). This inherent weakness coupled with maintenance burden of having to constantly update the signature databases has led to adoption of more intelligent behavior-based detections.

To get around these restrictions, scholars resorted to classical machine learning (ML) to come up with anomaly-based systems. Those models train on a model of normal network behavior and raise an alarm on large deviations as it can be an intrusion (Ahmed, Mahmood and Hu, 2016). Such algorithms as Support Vector Machine (SVM) or ensemble methods (like the Random Forest) were found to show significantly better results, especially in detecting the never-before-seen threats (Alshuaibi, Almaayah and Ali, 2021). Nevertheless, one of the shortcomings of these classic ML methods is that they heavily rely on the manual feature engineering process that is both time-consuming and demanding to be productive, as it requires deep domain knowledge (Mohammadi and Namadchian, 2021).

The advent of **deep learning (DL)** marked a paradigm shift, largely overcoming the feature engineering bottleneck. DL models, with their multi-layered neural network architectures, can perform automatic feature extraction, learning hierarchical representations directly from raw data (Kaur, Gabrijelcic and Klobucar, 2023). For network traffic analysis, which is inherently sequential, **Recurrent Neural Networks (RNNs)** and their variants have proven particularly effective. Standard RNNs, however, struggle with long-term dependencies due to the vanishing gradient problem. This led to the development of **Long Short-Term Memory (LSTM)** networks, which use a gating mechanism to regulate information flow, allowing them to remember relevant information over long periods (Hochreiter and Schmidhuber, 1997). This makes LSTMs exceptionally effective at detecting sophisticated attacks that unfold over extended timeframes (Kasongo and Sun, 2020).

The following table summarizes the key conclusions from influential papers in the field, illustrating this evolutionary trajectory.

Author(s) & Year	Focus / Methodology	Conclusion / Key Finding
Ahmed, M. et al. (2016)	Survey of anomaly detection techniques.	Provides a comprehensive overview of statistical and ML-based anomaly detection methods, establishing a baseline for NIDS research.
Buczak, A.L. & Guven, E. (2016)	Survey of ML/DM for cyber security.	Highlights that while ML shows promise, challenges like high false alarm rates and the need for labeled datasets remain significant hurdles for practical deployment.
Hochreiter, S. & Schmidhuber, J. (1997)	Introduction of LSTM.	Proposed the Long Short-Term Memory (LSTM) architecture to overcome the vanishing gradient

		problem in standard RNNs, enabling learning of long-term dependencies.
Sharafaldin, I. et al. (2018)	Introduction of the CICIDS2017 dataset.	Created a more realistic and comprehensive benchmark dataset to address the shortcomings of older datasets like KDD'99, which lack modern attack diversity.
Shone, N. et al. (2018)	Deep learning approach for NIDS.	Proposed a non-symmetric deep auto-encoder for unsupervised feature learning, demonstrating improved performance over traditional methods.
Khraisat, A. et al. (2019)	Survey of IDS techniques and datasets.	Reinforces that signature-based systems are ineffective against zero-day attacks and that anomaly detection, while powerful, struggles with high false-positive rates.
Al-Haj-Ali, H. et al. (2019)	Survey of ML-Based IDS.	Concludes that most IDS research focuses on accuracy metrics while overlooking real-world deployment challenges like scalability and resource consumption.
Aldweesh, A. et al. (2020)	Survey on deep learning for IoT IDS.	Reviews deep learning solutions for IoT security, emphasizing the need for lightweight models and addressing the gap between theoretical models and practical implementation.
Kasongo, S.M. & Sun, Y. (2020)	Deep learning with feature extraction for wireless IDS.	A wrapper-based feature selection method combined with an LSTM model improved detection rates, showing that hybrid approaches are highly effective.
Carzaniga, A. et al. (2021)	Survey on serverless computing.	Provides a thorough overview of the serverless paradigm, its benefits (scalability, cost), and challenges (cold starts, resource limits), establishing its context.
Mohammadi, S. & Namadchian, A. (2021)	A deep learning approach for NIDS.	Proposes a framework using CNN and GRU, reinforcing that deep learning models eliminate the need for manual feature engineering and achieve high accuracy.
Sayegh, H.R. et al. (2024)	LSTM with feature selection and SMOTE.	Demonstrated that an LSTM model, when combined with feature selection and techniques to handle class imbalance (SMOTE), achieves superior detection on CICIDS2017.

2.2 Synthesis and Identified Research Gap

The literature provides a clear trajectory in the evolution of NIDS, moving from static, signature-based systems to dynamic, intelligent models powered by deep learning. It is well-established that

for analyzing sequential network traffic data, LSTM networks are a highly effective architecture (Sayegh, Dong and Al-madani, 2024). The CICIDS2017 dataset is confirmed as a robust and relevant benchmark for training these models (Sharafaldin, Lashkari and Ghorbani, 2018).

In parallel, the field of **Machine Learning Operations (MLOps)** has matured to address the challenges of deploying models in production. MLOps emphasizes the entire lifecycle, from data ingestion to model monitoring, focusing on automation and scalability. Within this domain, **serverless computing** has emerged as a powerful paradigm, offering a highly scalable and cost-efficient alternative to traditional server-based deployment by abstracting away infrastructure management (Carzaniga et al., 2021).

However, a review of the existing research reveals a significant gap at the intersection of these domains. Many studies focus intensively on the theoretical performance and accuracy of LSTM models for intrusion detection (Aldweesh, Al-Azawei and Al-Zewairi, 2020). Yet, they often conclude with model evaluation metrics and fail to provide a detailed blueprint for a real-world, scalable deployment architecture. True to Al-Haj-Ali et al. (2019), the academic literature leaves a lot to be desired when it comes to real-world deployment drives such as resource utilization, scalability, and maintenance.

This project will fill that gap. It also covers the end-to-end-process in the delivery of a high-performance LSTM model on CICIDS2017 dataset and its deployment on a fully serverless, real-time and publicly accessible model architecture informed by MLOps principles. Due to the combination of the state of the art deep learning-based threat detection with a contemporary, scalable paradigm of operations, the given piece of research offers a holistic and feasible road map that makes a significant contribution to the challenge of cloud security by introducing an innovative, united solution.

3. Research Methodology

The given chapter reveals the methodology framework of this process with the idea to design, develop, and implement the proposed real-time serverless anomaly detection system. The methodology adheres to a constructive research design where it seeks to provide an intrusion detection pipeline that is fully usable because the pipeline remains cloud-native, as opposed to a purely theoretical model. The process can be thought of as a linear flow of work with each output being used as the input of the next stage ensuring that the result is replicable and logically sound all the way to the deployment. The major phases are:

- Draft of Research Design and Working Flow
- Data PreAcquisition and preprocessing
- Model building, and training
- Model Evaluation
- Serverless deployment and API Integration

The chapter is a chronological account of how the end-to-end, real-time anomaly detection system was designed, constructed and implemented. It describes the technical choices and practical decisions made at every step of the project, including data acquisition, and deployment of a publicly available machine learning model. The methodology provides a reproducible low technology workflow that is easy to follow and see a clear path to a production ready service.

This chapter will start with presenting a description of the constructive research design and workflow. It then transitions into the data handling minutiae that includes the process of selecting, ingesting, and heavy preprocessing of the CICIDS2017 dataset. The chapter, which follows it, outlines the architecture, training and assessment of the Long Short-Term Memory network (LSTM). Lastly, it records how to operationalize the model, how to bring it to a fully serverless implementation on Amazon Web Services (AWS) and put it in front as a live API endpoint to infer in real-time.

3.1 Research Design and Workflow

The research methodology used in this study is constructive in nature, that is, it targets the development of a new technical infrastructure: a complete and implemented, cloud-native intrusion detection system. The workflow was constructed in a sequential pipeline where the output of one stage becomes the input of other and makes the sequential logic achievable such that an output is a production service of the data.

The central steps of the working process are:

1. **Data Acquisition and Preprocessing:** Obtaining the CICIDS2017 dataset, storing it in a centralized cloud storage (Amazon S3) and subsequently cleaning, transforming, scaling to features and encoding it to be used in training the model.
2. **Model Development and Training:** Designing, training, and implementing a 1D LSTM neural network with the TensorFlow/ Keras platform to classify network traffic.
3. **Model Evaluation:** The performance of the trained model on a held-out test set should rigorously be evaluated using important measures of classification performance that can substantiate the effectiveness of the trained model.

API Integration and Deployment: Serializing the pipelines used to integrate the validated model and deploy to an end-to-end managed, serverless inference service (SageMaker Serverless Inference). A serverless function (AWS Lambda) will then wrap this service and then be made available as a publicly accessible HTTP endpoint through Amazon API Gateway.

Figure 3.1: Workflow Diagram

3.2 Data Acquisition and Preprocessing

3.2.1 Dataset Selection and Ingestion

The data collection of the foundation of this research is the CICIDS2017, which is chosen because it is regarded as modern and real-world benchmark to study NIDS research. This, as opposed to the older sets, is incredibly diverse with a range of multi-stage attacks of various types, as well as benign traffic, to train a strong classifier. The dataset which was spread across various CSV files was ingested and centralized in an Amazon S3 bucket. It was selected on the basis of scalability, durability and compatibility with the AWS ecosystem to be the single source of truth to the project.

3.2.2 Data Wrangling and Transformation

Data wrangling and preprocessing were done in an AWS SageMaker Notebook, which is an easily started managed Jupyter environment, having the key ML packages installed. We initially programmatically cleared the several CSV files in S3 into 1 panda DataFrame.
This was followed by a systematic data cleaning procedure. This included the elimination of unnecessary whitespace in column names, elimination of empty or faulty columns, and these were the numerical errors. One cleaning action that was critical was that of the presence of infinite values (+inf, -inf) which are computationally not feasible with most ML algorithms. It replaced these values with NaN indicators and the entire row in which a NaN occurred was dropped, guaranteeing the completion of the resultant dataset and resulting in a clean dataset, which would support the model stability.

3.2.3 Feature Engineering and Data Splitting

The given dataset was fine and the emphasis was laid on making it ready to go to the neural network. The column named as Label, which would define the classification such as BENIGN or DoS Hulk, was assigned as the target variable (y) and the other 78 numeric columns were converted into features (X).

The following three important preprocessing steps were then performed with the help of scikit-learn library:

1. Label Encoding: categorical string labels were encoded to integers with the help of LabelEncoder which is a prerequisite to train the model on target variable y.
2. Feature Scaling: StandardScaler was applied to W. This scales each of the features by shifting it to have a mean of zero and a standard deviation of one so that features that have a large range of possible values will not have an inordinate weight in the learning process.
3. Train-Test Split: The processed data were split into a training set (80%) and testing (20%) set by means of the train_test_split function. This decomposition is important in measuring the model potential to generalize to new, unseen data. There was a random_state that was initialized to make the split reproducible and deterministic across experiments.

3.3 LSTM Model Development

3.3.1 Model Architecture

As the main component of the detection system, a 1D Long Short-Term Memory (LSTM) network was selected, and it was implemented on the basis of the TensorFlow/Keras framework. LSTM architecture can describe a sequential data quite effectively, so it should work quite well to describe the temporal dependencies of the network traffic flows.
The Sequential model was constructed with the following layers:

- **Input Reshaping: A** first layer to convert 2D input data into three-dimensional tensor form needed by LSTM layers.
- **LSTM Layer:** It is the main processing layer with 64 memory units to learn the time patterns to distinguish between innocent and malicious traffic.
- **Dropout Layer:** A regularization layer with a rate of 0.3, random deactivation of fraction of neurons during training in order to prevent overfitting.
- **Output Layer:** A last Dense layer including a single, the neuron, and a sigmoid activity output. This gives a figure between 0 and 1 which is the likelihood that an input sample is anomalous.

3.4 Model Training and Evaluation

The model was compiled with the **'adam' optimizer** and **'binary_crossentropy'** as the loss function, standard choices for efficient binary classification. Training was conducted for 5 epochs with a batch size of 128, using 10% of the training data for validation after each epoch to monitor for overfitting. The model's final performance was then formally assessed on the completely unseen test set to provide an unbiased evaluation of its predictive accuracy.

3.5 Serverless Deployment and API Integration

3.5.1 Model Packaging and Deployment

After successful training, the Keras model was prepared for deployment. It was first saved in the versioned **TensorFlow SavedModel format**, which is required by AWS SageMaker's serving environment. This entire model directory was then compressed into a model.tar.gz archive and uploaded to the project's S3 bucket, creating the final deployment artifact.

For deployment, a serverless approach was chosen to maximize scalability and cost-efficiency. A ServerlessInferenceConfig was defined in SageMaker, specifying a memory size of 2048 MB and a maximum concurrency of 5 requests. Then the deploy() method was executed, which automatically created a SageMaker Serverless Inference endpoint. This enabled access to the trained LSTM model to be used in real-time predictions without the need to manage underlying servers.

3.5.2 API Integration for Real-Time Access

To make the model's predictions easily accessible, a public-facing HTTP API was created using a combination of AWS Lambda and Amazon API Gateway.

1. **AWS Lambda Wrapper:** A lightweight **AWS Lambda function** was created to act as a secure intermediary. Written in Python, this function's sole purpose is to receive a JSON payload from an API request, invoke the SageMaker endpoint with that payload, and return the model's prediction in the HTTP response.
2. **Amazon API Gateway:** An **HTTP API** was created in the API Gateway service. A POST /predict route was defined and integrated directly with the Lambda function. This automatically generated a public invoke URL, making the entire deep learning model accessible for real-time inference through a simple HTTP POST request. This fully decoupled and scalable architecture provides a robust and secure method for serving machine learning predictions.

Following such an organized approach, the research demonstrates that the research is traceable, scalable, and capable of being reproduced, which is critical to face practical purposes in enterprise cloud settings. The staged approach also promotes clear mapping between research objectives and technical decisions and evaluation metrics, making the systematic validation of the effectiveness of the system possible. Moreover, the inclusion of the serverless computing concepts in every deployment phase will make the resulting solution not only precise but also each in terms of operation efficiency and economical feasibility. Subsequent areas of this chapter go into closer detail about how each step of the methodology is derived, what justified the selection of tools, algorithms and architectures, and how each of those specific selection aids in the grander objective of delivering a robust real-time anomaly detection service.

4. Design Specification

The presented system is a modular, cloud-native system with real-time anomaly detection on network traffic based on deep learning and serverless computing. The design focuses on scalability, low latency inference, and cost-efficiency with the ease of integration into existing enterprise security infrastructures.

4.1 System Requirements

The functional requirements include, the capability to process and classify network traffic perceivably in real time. Proper identification of malicious patterns, including zero-day threats, at a low false positive rate. Automatic scaling to adjust to the varying degrees of traffic without the need to manually adjust it and integration with third-party monitoring and alerting via API.

Other requirements which are not functional:

- **Scalability**: Be able to support data streams in large volumes without performance attrition.
- **Latency**: Ensure that inference performance is under one second in real-time detection.
- **Reliability**: Make sure they are kept moving with as minimum time maintenance as possible.
- **Cost-efficiency**: Cloud resources optimisation in terms of reducing operational costs.

4.2 Architecture Overview

The architecture is in a five-stage pipeline:

Data Ingestion: The source of traffic data in the experiment will be the CICIDS2017 resource and live network tapping in the real world. The data is taken in by AWS Kinesis or similar streaming services.

Preprocessing Layer: A preprocessing engine runs on AWS Lambda to normalise and clean raw data, as well as transform it into feature vectors.

Model Inference Layer: A deep learning model (LSTM-based) is deployed as a serverless-function so that scalability is automatic because the height of the response is proportional to the number of incoming requests.

Integration with the API Gateway: AWS API Gateway adds client applications access to the detection service with authentication and rate-limiting rules.

Alerting & Logging: The findings are sent to a cloud-based monitoring dashboard and recorded in AWS S3 to be revisited in case of audit and retraining.
This architecture makes all the components loosely coupled so that they could be upgraded independently or replaced with any other components without disturbing the system.

4.3 Stack Used

- **Programming:** Python (TensorFlow/Keras to implement the deep learning model).
- **Cloud Services:** AWS lambda, API gateway, S3, Kinesis and cloud watch.
- **Model Deployment Framework:** TensorFlow Serving within a serverless setting.
- **Data Processing:** Pandas and Scikit-learn will be used in preprocessing the data.

4.4 Explanation of design decision

To support pay-per-use cost models and minimize overhead in managing infrastructure, a serverless architecture was selected. LSTM networks were used because they provide good results

in sequential dependencies of the patterns in the network traffic, which facilitates complex attacks detection. Cloud-native services will have the characteristic of it being able to scale elastically, have high availability, and be integrated with enterprise security tools.

The given design specification offers a transparent roadmap of realisation by which the proposed system will fulfil both research and practical deployment purposes.

5. Implementation

This chapter provides a detailed, practical account of the execution of the methodology outlined in Chapter 3. It translates the architectural design into a concrete narrative of the project's development, covering the specific tools, code, and configurations used to build the real-time anomaly detection systemThe aim is to provide an easy to follow, step-by-step tutorial of how this was actually done, not only in terms of initially setting up the development environment but also in getting that live, demonstrated API up and running and critically evaluated.
The initial part of the discussion proceeds with a description of the development environment and the set of technologies that were the basis of the project. It is then moved onto the most important stages of implementation: data preparation, model building and training. The next sequential order is given to the detailed assessment of the model performance with reference to the details of its implementation. This involves analysis of a naive classifier to give a baseline, evaluation of key metrics of the final LSTM model and a comparative analysis against a conventional machine learning model. Last, the chapter lists deployment of the model into production-ready, serverless pipeline, giving a comprehensive picture of the project, all the way back to its conception.

5.1 Development Environment and Tooling

The whole project was designed and implemented under the Amazon Web Services (AWS) cloud ecosystem and incorporates use of a series of managed services to guarantee a scalable, secure, and efficient workflow. The selection of a cloud-native environment was a strategic decision to mirror real-world enterprise conditions and to harness the power of managed services to accelerate development.

The primary development environment was an **AWS SageMaker Notebook instance**, which provided a fully managed Jupyter Notebook environment. This choice eliminated local setup complexities related to dependency management and hardware constraints, while offering seamless, low-latency integration with other essential AWS services. The key components of the technical environment were:

- **Amazon S3 (Simple Storage Service):** Served as the central, scalable data lake for the project. It was used to store the raw, multi-file CICIDS2017 dataset and later to host the final packaged model artifact (model.tar.gz) required for deployment.
- **AWS SageMaker:** This service was the cornerstone of the entire development and deployment process. A SageMaker Notebook was used for all interactive coding, from data cleaning and preprocessing to model training and evaluation. For deployment, **SageMaker Serverless Inference** was utilized to host the model without managing any underlying server infrastructure, a key component of the project's objective.

- **AWS Lambda & Amazon API Gateway:** These two services formed the serverless application layer responsible for exposing the model's predictive power. Lambda was used to create a lightweight wrapper function that invokes the SageMaker endpoint, while API Gateway exposed this function as a secure, public HTTP API.
- **Core Python Libraries:** The implementation relied on a set of industry-standard, open-source Python libraries:
    - **Pandas & NumPy:** Used extensively for high-performance data manipulation, cleaning, and efficient numerical operations.
    - **Scikit-learn:** Provided essential tools for preprocessing, including LabelEncoder for converting categorical targets into a numerical format, StandardScaler for feature normalization, and the train_test_split function for partitioning the data.
    - **TensorFlow with Keras:** The deep learning framework of choice, used for building, compiling, and training the LSTM model with its high-level, user-friendly API.
    - **Boto3:** The official AWS SDK for Python, used within the notebook to programmatically interact with S3 and other AWS services.

5.2 Data Ingestion and Preparation

The implementation began with the critical task of ingesting and consolidating the multiple CSV files of the CICIDS2017 dataset. This dataset, chosen for its realism and inclusion of modern network attacks, is distributed across several files. Using the boto3 library within the SageMaker Notebook, the script programmatically listed all dataset files stored in the designated S3 bucket. Each CSV file is then read into temporary by a loop pandas DataFrame and combined to a single (unified) DataFrame, called merged_df. After that, a structured but time-consuming data cleaning procedure was started. The leading and trailing whitespace was removed in all the column headers so that an error does not occur in the process of selecting features. Numerical inconsistencies were also addressed in the script, when the script detected an occurrence of both positive infinity and positive minus infinity it replaced it with the NaN (Not a Number) representation in NumPy. Lastly, in order to make the data fed into the model be of good quality, NaN values were removed by dropping rows in the DataFrame. This is essential in model training stability and avoiding some runtime errors. On the preliminary examination of the 'Label' column, it became apparent that the ratio of the classes was very heavily skewed, with benign traffic preponderating against cases of malicious traffic. This is a major feature of realistic network.

5.3 Preprocessing and Feature Scaling

Having a clean and unified DataFrame, the second step was dedicated to transforming the data in such a way that they could be used with the LSTM model. First, the DataFrame was separated into features (X) and the target label (y).

As the target, the Label column was selected and the rest 78 numerical columns formed the feature set after that preprocessing steps (which are necessary to allow most machine learning algorithms to work correctly) were applied with the help of scikit-learn:

1. Label Encoding: The discrete string types in the target variable y (e.g. BENIGN, DoS Hulk) were changed to a numeric representation where LabelEncoder was used. This gave a different series y enc where unique class labels were represented by corresponding integer, a requirement needed in loss functions calculation in the model.
2. **Feature Scaling:** offered the StandardScaler, applied to the feature set X. The mean of every feature was made 0 and a standard deviation of 1 in this important step that made the data normal. It does not allow large-range-valued features to disproportionately affect the weight updates the model is performing during training and makes the optimization algorithm converge faster.
3. **Train-Test Split:** Finally, the preprocessed data was partitioned into training (80%) and testing (20%) subsets using the train_test_split function. This division is fundamental for a valid model evaluation, as it allows the model's performance to be tested on a hold-out set of data it has never seen before. To ensure the results were reproducible across experiments, the random_state was set to a fixed value of 42.

5.4 LSTM Model Implementation and Training

The core deep learning model was implemented using the high-level
Sequential API from TensorFlow/Keras, which allows for the intuitive, layer-by-layer construction of a neural network. The architecture was designed as follows:

- An input **Reshape** layer was used to transform the 2D input data (samples, features) into the 3D tensor format (samples, timesteps, features) that is required by all recurrent layers in Keras, including LSTMs.
- An **LSTM** layer with 64 hidden units served as the main processing component of the network. This layer is designed to learn temporal dependencies and long-range patterns in the sequential network data.
- A **Dropout** layer with a rate of 0.3 was included immediately after the LSTM layer. This is a critical regularization technique that helps prevent overfitting by randomly deactivating 30% of the neurons during each training step, forcing the network to learn more robust features.
- A **Dense** output layer with a single neuron and a 'sigmoid' activation function produced the final classification. Since the sigmoid function returns a value between 0 and 1, it is perfect in cases where it is required to solve binary classification issues such as the one in consideration.

It used the compile() method to set a model to train. It used the optimizer known as adam, and the loss used is known as binary_crossentropy which are both default, and highly efficient choices in binary classification problems. Training was done by invoking the .fit() method on training data 5 times with a batch size of 128.

6. Evaluation

Analysis of model performance is very important in determining the effectiveness of model and areas of ineffectiveness. The complex approach was made with the use of the baseline that led to the detailed description of the final model.

6.1 Baseline Performance: A Naive Classifier

In order to create a benchmark of working with a performance indicator and emphasize the issues of the unbalanced CICIDS2017 dataset, a naive binary classifier was tested. This model proved to be very precise with an accuracy of 1.0 and a very low recall of 0.0901. This result indicates that the model simply learned to predict the majority class (benign traffic) for almost every input, thereby failing to identify most of the actual anomalies. This underscores the inadequacy of accuracy alone as a metric for imbalanced datasets and justifies the use of more nuanced metrics like the F1-score for the final model evaluation.

Metric	Score
Precision	1.0000
Recall	0.0901
F1 Score	0.1654

Table 1: Performance of a Naive Binary Classifier

6.2 Final Model Performance Analysis

The performance of the final trained LSTM model was formally measured by calling model.evaluate() on the held-out test data. A comparative analysis was also conducted against a traditional RandomForest classifier to benchmark the deep learning approach. The results, shown in Table 2, demonstrate the exceptional performance of the LSTM model.

Model	Split (Train/Test)	Accuracy	Macro_F1 Score		
RandomForest	70/30	0.999889	0.999645		
RandomForest	60/40	0.999889	0.999645		
LSTM	**70/30**		**0.999593**		**0.9987**
LSTM	**60/40**		**0.999333**		**0.997877**

Table 2: Performance Comparison of Final Models

The LSTM model achieved an accuracy and Macro F1-Score well above 99.9%, indicating its strong ability to correctly classify both benign and malicious traffic. While the RandomForest model posted slightly higher scores, the selection of the LSTM model for deployment was based on its inherent architectural advantages for handling sequential data and its ability to learn features automatically, making it theoretically more robust against novel, zero-day attacks that may have different temporal patterns than those seen in the training set.

6.2.1 Serverless Deployment and API Implementation

The final phase of the implementation involved deploying the packaged model and exposing it as a public API, thereby transforming it from a static file into a live, operational service.

6.2.2 Model Export and Packaging

To be deployed on AWS SageMaker, the trained Keras model had to be packaged into a specific format. This was accomplished using a series of commands within the notebook. The model was first re-saved in the versioned
TensorFlow SavedModel format inside a directory structure (model_dir/1/). This directory was then compressed into a model.tar.gz archive, which is the final deployment artifact containing the model's architecture, weights, and serving signature.

6.2.3 Endpoint Deployment and API Integration

The deployment process was executed programmatically from the SageMaker Notebook.

1. **SageMaker Serverless Endpoint Deployment:** The model.tar.gz artifact was uploaded to S3. A TensorFlowModel object was instantiated in the SageMaker SDK, pointing to the model's S3 URI. A ServerlessInferenceConfig was defined to configure the deployment with 2048 MB of memory and a maximum concurrency of 5 requests. The deployment was initiated by calling tf_model.deploy(), which provisioned a live, invocable SageMaker endpoint named anomaly-detect-sls.
2. **AWS Lambda Wrapper Function:** An AWS Lambda function named invoke-anomaly-endpoint was created to serve as a secure intermediary between the public internet and the SageMaker endpoint. Configured with the Python 3.9 runtime, its handler code was implemented to receive a JSON payload, use the boto3 client to invoke the endpoint, and return the model's prediction.
3. **Amazon API Gateway Configuration:** To make the service publicly accessible, an HTTP API was created in Amazon API Gateway. A POST route at the path /predict was configured and integrated directly with the Lambda function. Upon deployment, a public invoke URL was generated, becoming the final entry point for the real-time anomaly detection service, ready to accept prediction requests from any standard HTTP client.

6.4 Discussion

From the evaluation metrics we can see that the LSTM model out performs the baseline Random Forest model by having the F1-scores exceeding 99.9% in the classification of the network traffic. This deep learning architecture was selected in particular because of its strength in modeling sequential data making it exceptional in identifying complex and other minority attacks that other models might miss. By the deployment of the model using SageMaker Serverless Interface it provided a powerful pattern for the ML-driven security, which offers so many advantages in comparison with the traditional deployments. By this deployment even if there is a spike in the network traffic it will be handled automatically because of its auto scaling capability.

This model performs only binary classification, with the help of a multi class classifier the specific type of anomaly can be detected which could help in providing more insight to security operations teams by enabling them to make alerts and effective incident responses. The future works should also focus on improving the cold start latency in serverless platforms.

7. Conclusion

The significant results of the research have been generalised and discussed in this chapter, which also evaluates how well the research objectives have been achieved and what possible future ways to improve the proposed system may be. This research successfully achieved its primary objective: the development and deployment of a scalable, real-time intrusion and anomaly detectioIn system specifically designed to the special requirements of contemporary cloud security environments. At a time when the high complexity of the cloud and the ever-changing nature of the cyber-threat landscape tend to surpass the evolving security principles previously in use, the practical and successful combination of deep learning sophistication and a fully spun-up serverless, event-driven design on Amazon Web Services can be considered as the distinct advantage of the current project. The piece will be filling one of the most important missing research links that shifts the concept of merely reviewing the theoretical models to a technical roadmap step by step showing how an actual problem-based system can be built that can be in operation. This way, this paper has shown that the issue of high detection accuracy and operational efficiency in contemporary cybersecurity has an effective and strong means to solve the twofold challenge. The implementation at the end that resulted in a publicly available API that can provide real time predictions is the final validation of the end-to-end feasibility of the proposed architecture and its useful practical application.

The systematic and meticulous methodological path taken was outlined by the ingestion and careful preparation of the complicated, multi-file CICIDS2017 dataset. Considering the issues presented by the dataset, namely its size and class imbalance, the first phase consisted of an arduous cleaning and preprocessing pipeline. It included label encoding of the target variable and feature scaling.

To ensure the stability and reliability of further modeling step it is normalising data using StandardScaler. The core of the system has been a 1D Long Short-Term Memory (LSTM) neural network model created, and trained via the Python framework TensorFlow/Keras. This particular architecture of deep learning has been selected due to its successfulness in sequential modeling aspect that it has shown, and thus it would be highly effective in capturing the temporal aspect of network traffic in an attempt to detect low and slow, complex attacks, otherwise overlooked by other models. The resulting model was as shown, state of art in its efficacy with an accuracy and F1-scores above 99.9% in classifying network traffic.

However, the core innovation of this research was realized in the deployment pipeline. The trained model was packaged into a serverless-compatible artifact and deployed using **SageMaker Serverless Inference**. This endpoint was then fronted by a lightweight **AWS Lambda** function and exposed to the public via **Amazon API Gateway**. This serverless architecture provides a powerful paradigm for ML-driven security, offering tangible benefits that directly address the

operational challenges of traditional deployments. It provides true automatic scaling to handle unpredictable traffic spikes without manual intervention and operates on a pure pay-per-use cost model, which completely eliminates the financial burden of paying for idle, provisioned resources.

Despite the successful implementation, this research also illuminates several promising avenues for future work. The current model performs binary classification; a natural and valuable extension would be to develop a multi-class classifier capable of identifying the specific type sof anomaly (e.g., 'DDoS', 'PortScan', 'Botnet'). This would provide far more granular and actionable insights for security operations teams, enabling them to prioritize alerts and orchestrate more targeted and effective incident responses. Further investigation should also focus on mitigating the "cold start" latency inherent in serverless platforms. While acceptable for many use cases, this initial delay upon first invocation could be a limitation for applications with strict sub-second latency requirements. Investigating the ease such as the AWS provisioner concurrency to maintain instances of functions warm at an added price would be a helpful training to execute in exploring the balance amid performance and effective use.

Lastly, in order to guarantee efficiency over an extended period of time and counteract model drift- the fact that models performance will naturally decline over time once new, unseen attack vectors are introduced- the current architecture can be completed with an entirely auto MLOps pipeline that would allow constant model improvement. An example of such a pipeline would be used to automatically retrain the model over time on new data through a scheduled retrain, or an automatic retrain when the expected predictions are no longer being met with the current model. The latter would include developing an efficient feedback loop in the case of labeling new threats and their integration into the training set to make sure that the model improves as the threat environment does.

Finally, this project presents an effective, feasible guide to developing and implementing smart, cloud native security. It is effective in filling the gap in the literature between theoretical deep learning models and their implementation in the real world that has a practical and scalable nature, addressing an apparent need. The serverless strategy shown here is not only a technical decision; it is a strategic enabler which is extremely cost effective, scalable and helps in overcoming operational overhead significantly, enabling organizations to concentrate on security logic, not managing infrastructure. The book is an informative and practical source of reference to practitioners and researchers wishing to harness the sheer strength of computing serverless technology and artificial intelligence to tackle the multifaceted and constantly changing problematic issues of cybersecurity.

References

Ahmed, M., Mahmood, A.N. and Hu, J., 2016. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 60, pp.19-31.

Al-Haj-Ali, H., et al., 2019. A Survey of Machine Learning-Based Intrusion Detection Systems. In *2019 International Conference on Electronics, Communications and Information Technology (ICECIT)* (pp. 130-135).

Aldweesh, A., Al-Azawei, A. and Al-Zewairi, M., 2020. A comprehensive survey on deep learning for IoT intrusion detection systems. *Journal of King Saud University-Computer and Information Sciences*, 34(10), pp.8676-8698.

Alshuaibi, A., Almaayah, M. and Ali, A., 2021. Machine Learning for Cybersecurity Issues: A systematic Review. *Journal of Cyber Security and Risk Auditing*, 1(1), pp.36–46.

Buczak, A.L. and Guven, E., 2016. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2), pp.1153-1176.

Carzaniga, A., et al., 2021. A survey on serverless computing. *ACM Computing Surveys (CSUR)*, 54(7), pp.1-36.

Hochreiter, S. and Schmidhuber, J., 1997. Long short-term memory. *Neural computation*, 9(8), pp.1735-1780.

Kasongo, S.M. and Sun, Y., 2020. A deep learning method with wrapper based feature extraction for wireless intrusion detection system. *Computer Networks*, 168, p.107022.

Kaur, R., Gabrijelcic, D. and Klobucar, T., 2023. Artificial intelligence for cybersecurity: Literature review and future directions. *Information Fusion*, 97, p.101804.

Khraisat, A., et al., 2019. Survey of intrusion detection systems: techniques, datasets and challenges. *Cybersecurity*, 2(1), pp.1-22.

Mohammadi, S. and Namadchian, A., 2021. A deep learning approach for network intrusion detection. *Journal of Cloud Computing*, 10(1), pp.1-15.

Sayegh, H.R., Dong, W. and Al-madani, A.M., 2024. Enhanced Intrusion Detection with LSTM-Based Model, Feature Selection, and SMOTE for Imbalanced Data. *Applied Sciences*, 14(2), p.479.

Sharafaldin, I., Lashkari, A.H. and Ghorbani, A.A., 2018. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP)* (pp. 108-116).

Shone, N., et al., 2018. A deep learning approach to network intrusion detection. *IEEE transactions on dependable and secure computing*, 16(5), pp.837-846.