National College of Ireland

Configuration Manual

MSCCLOUD Research Project
Master of Science in Cloud Computing

Asfand
Student ID: 23358530

School of Computing
National College of Ireland

Supervisor: Vikas Sahni

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Asfand
Student ID:	23358530
Programme:	Master of Science in Cloud Computing
Year:	2024-2025
Module:	MSCCLOUD Research Project
Supervisor:	Vikas Sahni
Submission Due Date:	11/08/2025
Project Title:	Configuration Manual
Word Count:	1668
Page Count:	11

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:	Asfand
Date:	10th August 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):	

Configuration Manual

Asfand
23358530

1 Introduction

1.1 Purpose of the Configuration Manual

The purpose of this configuration manual is to provide detailed technical guidance for installing, configuring, and deploying the AI-based credit card fraud detection system. This document ensures reproducibility of results and smooth setup for future deployment, integration, or research purposes.

1.2 Scope of the Document

This manual covers all stages of the setup, including the local development environment on Kaggle, model training and evaluation, model serialization, and deployment on AWS SageMaker. It includes library installation, dataset handling, environment configuration, and endpoint setup for real-time inference. It also describes the use of FastAPI (Ramírez (2023)) and Render (Render Team (2024))for exposing the inference endpoint externally.

1.3 Target Audience

This document is intended for someone who wants to replicate that can also include:

- AI/ML practitioners and researchers
- System integrators and software developers
- Academic supervisors and examiners evaluating the project
- Clients or stakeholders managing deployment via cloud infrastructure

1.4 System Overview

The system is an AI-based fraud detection solution trained on the Kaggle credit card dataset. The model leverages XGBoost in conjunction with SMOTEENN (Batista et al. (2004))for class imbalance handling. It is capable of identifying fraudulent transactions with high accuracy and low false positives. The model is trained and evaluated in a Jupyter Notebook environment on Kaggle, and deployed to Amazon SageMaker using a serialized '.pkl' artifact. An optional FastAPI server was developed for simplified communication with the deployed endpoint, hosted using Render.

2 System Requirements

This section outlines the essential hardware, software, and network prerequisites required for installing, running, and deploying the Credit Card Fraud Detection System. Meeting these requirements ensures seamless execution from local testing on Kaggle to final deployment on AWS SageMaker and Render.

2.1 Hardware Requirements

The system does not require high-end computing hardware for experimentation or model training. However, the following are my system specifications under which I have worked:

- **RAM:** 16 GB
- **CPU:** Core(TM) i5-8th Generation (4 Cores and 8 Logical Processors)
- **Disk Space:** 256GB

2.2 Software Requirements

The following software tools, libraries, and environments are required to run and deploy the project:

- **Operating System:** Windows 11 Pro Insider Preview 64-bit (10.0, Build 26100)
- **Python:** Version 3.10.12
- **Libraries:**
  - `pandas==1.5.3`, `numpy==1.24.3`, `matplotlib==3.7.1`, `seaborn==0.12.2`
  - `scikit-learn==1.2.2` (Pedregosa et al.; 2011), `xgboost==1.7.6` (Chen and Guestrin; 2016) , `lightgbm ==3.3.5` (Ke et al.; 2017)
  - `imblearn==0.10.1` (Lemaitre et al.; 2017), `joblib==1.3.2`, `fastapi==0.100.1`, `uvicorn==0.23.2`
  - `boto3==1.28.3`, `sagemaker==2.167.0` for deployment on AWS
- **Tools:**
  - `Jupyer Note Book` (for model training and evaluation)
  - `Render.com` account (for FastAPI frontend)
  - `AWS IAM` credentials

All dependencies are listed in the `requirements.txt` file included in the project repository.

2.3 Network Requirements

The project requires an active internet connection for the following tasks:

- Downloading Python packages and dependencies
- Accessing the Kaggle runtime environment and datasets
- Uploading model artefacts to AWS S3
- Deploying and interacting with SageMaker endpoints
- Hosting FastAPI app on Render and sending requests to model API

For AWS deployment Amazon Web Services (2023), ensure that appropriate permissions and policies are enabled for the IAM user to access S3, SageMaker, and endpoint services.

3 Installation Instructions

This section outlines the step-by-step process to set up the environment, run the fraud detection pipeline on Kaggle, and prepare the model for deployment using AWS SageMaker and Render.

3.1 Prerequisites

Ensure access to the following tools and platforms before proceeding:

- Kaggle account for model development and training
- AWS IAM credentials for SageMaker access
- Render account for FastAPI server deployment (optional)
- Internet connection to install required libraries

The required Python libraries can be installed using the following command in Kaggle or a local environment:

```
!pip install pandas
!pip install numpy
!pip install matplotlib
!pip install seaborn
!pip install scikit-learn
!pip install xgboost
!pip install lightgbm
!pip install fastapi
!pip install uvicorn
!pip install joblib
!pip install boto3
!pip install sagemaker
!pip install scikit-learn==1.3.2 imbalanced-learn==0.11.0
```

Note: Version `scikit-learn==1.3.2 imbalanced-learn==0.11.0` is specified to avoid compatibility issues encountered with more recent versions.

3.2 Installation Steps

The model was developed and tested using Kaggle. The following steps were taken:

1. Log in to Kaggle and create a new Notebook.
2. Upload the dataset via the **Add Data** panel in the Kaggle notebook interface.

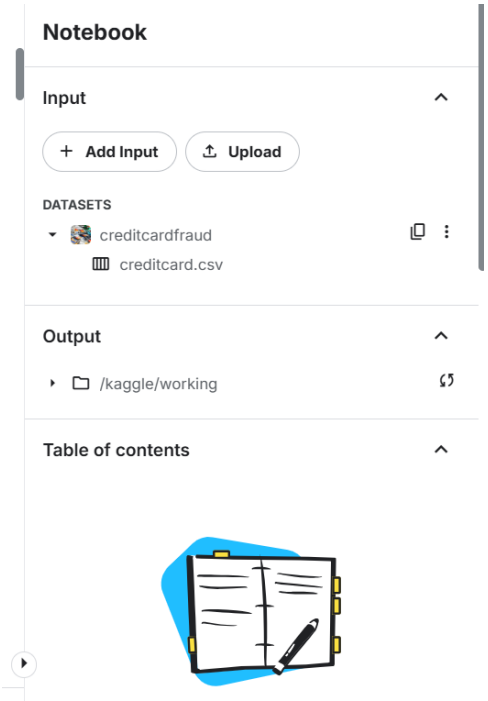

Figure 1: Uploading dataset to Kaggle

3. Run the provided code to perform data preprocessing, feature scaling, model training, and evaluation.

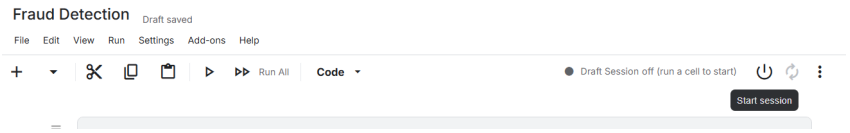

Figure 2: Running the model training and evaluation pipeline

4. After successful model training, serialize the model using Developers (2023):

```
import joblib
joblib.dump(model, 'fraud_detection_pipeline.pkl')
```

5. Download the file `fraud_detection_pipeline.pkl` for deployment purposes.

3.3 Environment Setup

A structured folder setup is maintained for deployment:

- `fraud_detection_pipeline.pkl` — serialized trained model
- `server.py` — FastAPI script for forwarding client requests to SageMaker endpoint
- `requirements.txt` — list of Python dependencies
- `code/inference.py` — SageMaker-compatible inference handler
- `model.tar.gz` — archived model artefact containing `.pkl`, `inference.py`, and `requirements.txt`

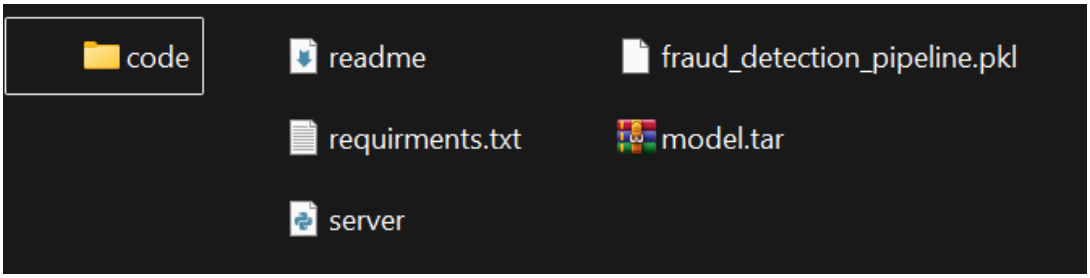

Figure 3: Environment Setup

If API tokens or secret keys are needed, environment variables can be set manually on Render or AWS using their console interfaces. This setup eliminates the need for local configuration files such as `.env`.

4 Model Deployment

This section outlines the deployment of the trained fraud detection model to AWS SageMaker for real-time inference. The objective was to ensure a reliable, scalable, and production-ready environment where predictions can be made via HTTP endpoints.

4.1 Deployment Strategy

The deployment was performed using the IAM user, ensuring all configurations and resources are stored within the AWS account. The approach utilized SageMaker's built-in support for Scikit-learn, combined with Amazon S3 for model storage and FastAPI for endpoint integration.

4.2 Step-by-Step Deployment Process

1. **Model Packaging:** The trained model was serialized into a `.pkl` file. A `model.tar.gz` archive was created containing the following structure:
   - Root directory: `fraud_detection_pipeline.pkl`
   - `/code` directory: `inference.py` and `requirements.txt`

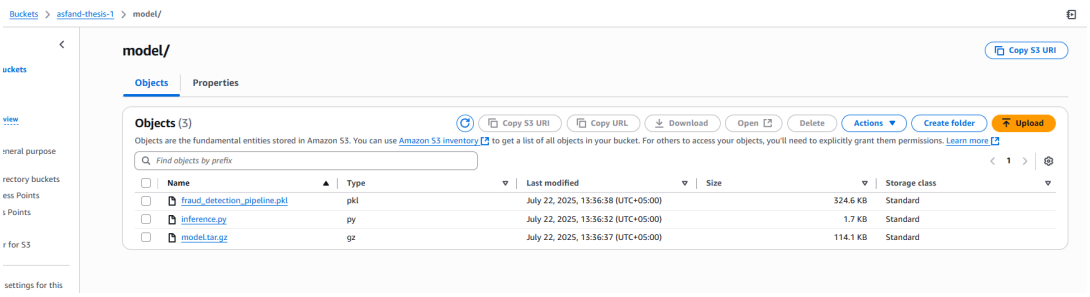

Figure 4: Model files uploaded to S3 bucket "asfand-thesis-1"

2. **Creating the Model on SageMaker:** The packaged model was uploaded to the S3 bucket and a new SageMaker model was created using the Scikit-learn container. Default configurations were used, including IAM execution roles and default environment variables.

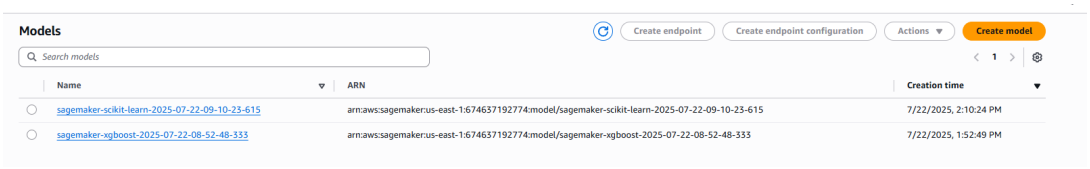

Figure 5: SageMaker Model Creation with Scikit-learn Image

3. **Endpoint Configuration:** A new endpoint configuration was defined with the model ARN, specifying a minimal instance (ml.t2.medium) to minimize costs. No training was performed in this environment, as the pre-trained model was directly deployed.

Figure 6: SageMaker Endpoint Configuration

4. **Deploying the Endpoint:** The endpoint named `fraud-detector-endpoint-v8` was deployed. It became operational in approximately 8 minutes. Monitoring and logs were available for CPU and memory utilization.

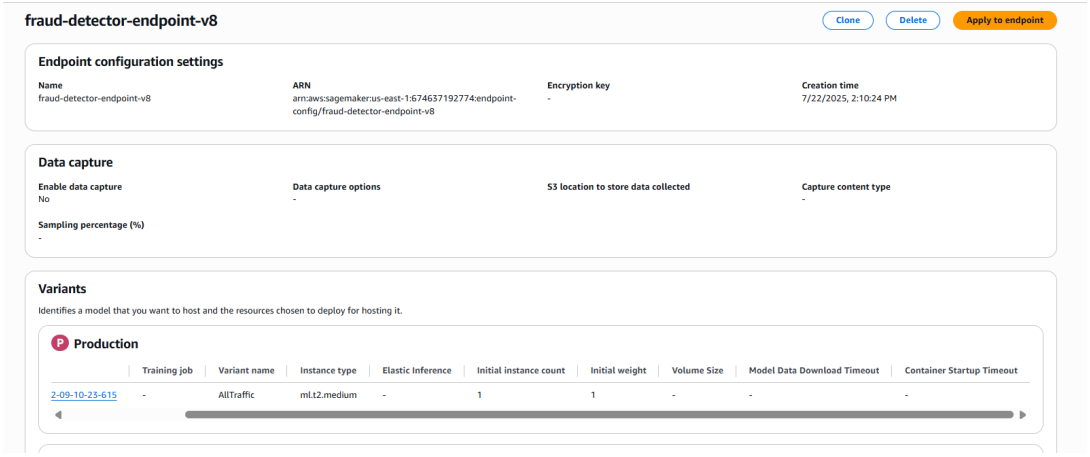

Figure 7: List of Deployed Endpoints

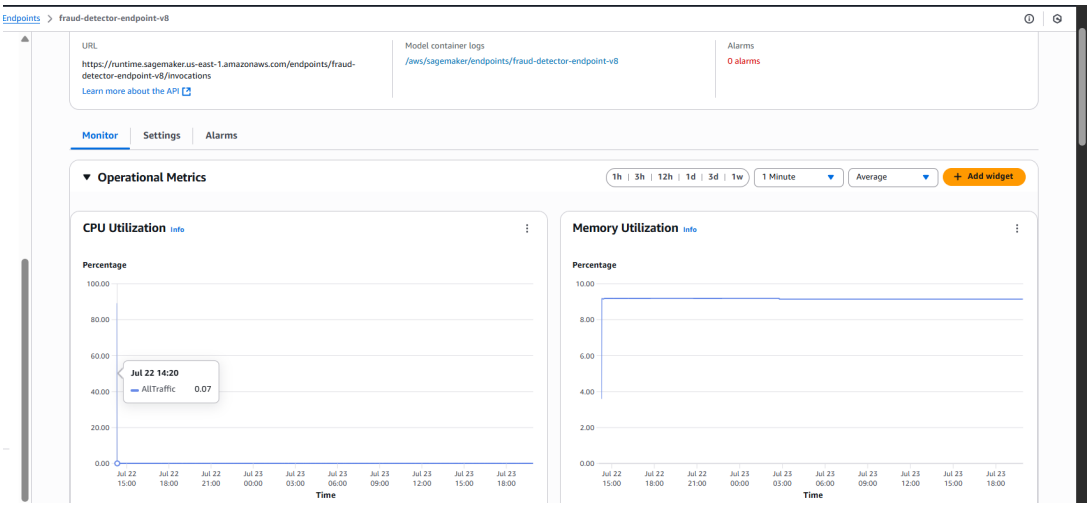

Figure 8: Monitoring Metrics for the Endpoint

5. **Integration with FastAPI:** To simplify usage, a FastAPI server was created that sends JSON-formatted data to the endpoint and returns predictions. This acts as a middleware between any frontend or testing environment and the SageMaker endpoint.

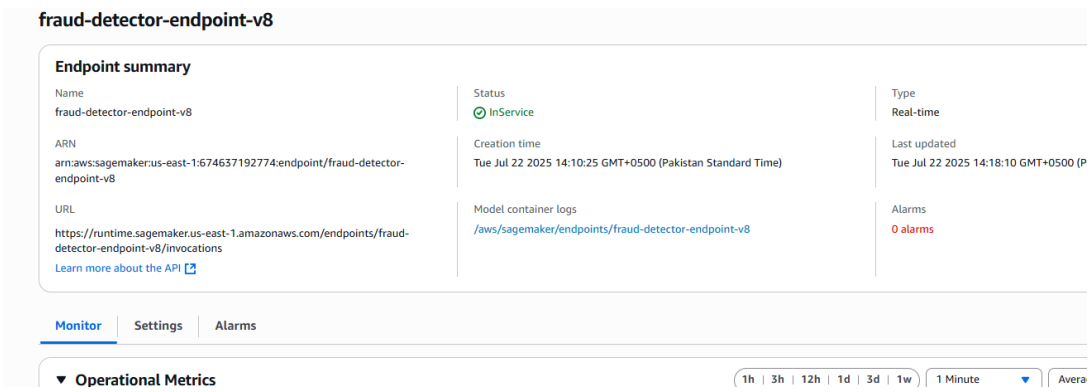

Figure 9: Endpoint Summary Showing URL and Status

4.3 Conclusion

This deployment ensures secure, real-time inference capabilities for fraud detection, with full logging and monitoring capabilities via SageMaker. The architecture allows for decoupled integration with applications through the FastAPI layer, while maintaining flexibility and low operational cost.

5 Running the Application

This section outlines how to execute and interact with the fraud detection system. The model is deployed using AWS SageMaker, while a FastAPI server acts as an interface, allowing users to send data for predictions via a RESTful API hosted on Render.

Initial development and testing were done in the Kaggle environment using Jupyter notebooks and Python scripts.

5.1 Production via Render and FastAPI

For production, a FastAPI server was deployed to Render. This server handles communication with the AWS SageMaker endpoint and exposes the model as a public API.

1. **Live API Documentation:** Users can interact with the API directly using the Swagger UI available at:

   `https://fraud-detection-tncm.onrender.com/docs`

   This UI provides documentation and allows you to send test requests via browser.

2. **API Usage:** Send a POST request to:

   `https://fraud-detection-tncm.onrender.com/predict`

   with input JSON formatted as:

```
{
  "input_data": [
    [0.5, -1.2, 0.3, ..., 1.7]
  ]
}
```

Figure 10: Endpoint Summary Showing URL and Status

3. **Response:** The API responds with the fraud prediction and its probability score.

4. **Backend Logic:** The FastAPI server forwards the request to the SageMaker endpoint, waits for the response, and returns it to the client.

5.2 Deployment Notes

- Render services pause inactive containers. The first request may take up to 50 seconds to activate the server. It's advisable to perform a health check prior to a demo.

- Logs and performance metrics can be accessed via the Render dashboard for debugging or optimization.

This configuration allows the model to be consumed via an intuitive, browser-accessible API interface with minimal setup on the client side.

References

Amazon Web Services (2023). Boto3 documentation, `https://boto3.amazonaws.com/v1/documentation/api/latest/index.html`.

Batista, G. E. A. P. A., Bazzan, A. L. C. and Monard, M. C. (2004). A study of the behavior of several methods for balancing machine learning training data, *ACM SIGKDD Explorations Newsletter*, Vol. 6, pp. 20–29.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794.

Developers, J. (2023). Joblib: running python functions as pipeline jobs, `https://joblib.readthedocs.io/en/latest/`.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q. and Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree, *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3146–3154.

Lemaitre, G., Nogueira, F. and Aridas, C. K. (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning, *Journal of Machine Learning Research* **18**(17): 1–5.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E. (2011). Scikit-learn: Machine learning in python, *Journal of Machine Learning Research* **12**: 2825–2830.

Ramírez, S. (2023). Fastapi: Web framework for building apis with python 3.6+ based on standard python type hints, `https://fastapi.tiangolo.com/`.

Render Team (2024). Render: Cloud hosting for developers, `https://render.com`.