DSA4266: Detection of GPS Spoofing in GOJEK

Group 1 GitHub Repository

A Muhammed Madhih
National University of Singapore
e0684508@u.nus.edu

Claudeon Reinard Susanto
National University of Singapore
claudeon.susanto@u.nus.edu

Lee Zheng Yao, Daniel
National University of Singapore
lzydaniel@u.nus.edu

Shonn Tan
National University of Singapore
e0725334@u.nus.edu

Tan Ze Jian
National University of Singapore
zejiantan@u.nus.edu

***Abstract*—The prevalence of Fake GPS usage among drivers in the GOJEK application poses integrity threats and privacy concerns, necessitating the development of robust machine learning models for anomaly detection. This study focuses on classifying trips as genuine or involving Fake GPS usage based on driver PING behavior data. Leveraging the GOJEK GPS dataset, we employ various machine learning techniques, including logistic regression, XGBoost, temporal convolutional neural networks (TCNN), long-short term memory (LSTM), and Autoencoders (AE), to build classification models. Hyperparameter tuning is performed to optimize model performance, measured by binary cross-entropy loss and test accuracy. Additionally, we conduct error analysis to identify model weaknesses and refine our approach.**

I. Problem Statement

The presence of drivers using Fake GPS in the GOJEK application poses a significant challenge, creating an unfair environment for honest drivers and potentially impacting the overall service quality for customers. This scenario introduces integrity threats, as the manipulation of location data undermines the trustworthiness of the service. Additionally, the issue raises concerns about data privacy, emphasizing the ethical and legal aspects of safeguarding location information. In many instances, location information is used as valuable information to solve many crimes.

As a cybersecurity measure, the task is to develop a machine learning model that can accurately classify trips where Fake GPS is being used based on the PING behavior of drivers performing an anomaly detection. This will help to ensure a secure and reliable platform for both drivers and users.

II. Dataset

We will be using the GOJEK GPS dataset from Kaggle [1] with 567,545 data points and 11 columns (Table V) to distinguish between genuine and fake GPS by identifying patterns indicative of each. The dataset comprises of anonymous driver's PING signals and 3,500 unique *order_id* column which serves as a unique identifier for each ride session. It contains nine feature variables and one binary target column, *label*, to determine whether it is spoofed (label 0) or not spoofed (label 1).

This can be achieved through analysis of features to uncover characteristics unique to genuine GPS signals and those associated with fake GPS usage. Features such as signal strength, consistency of signal reception, geographical accuracy, and temporal patterns may exhibit differences between genuine and fake GPS data.

III. Prior Works

Previous works on the GOJEK GPS dataset from Kaggle primarily utilized classical machine learning models like XGBoost to analyze the data. However, these studies notably lacked implementation of neural network models. Moreover, the reported results often focused solely on accuracy metrics, neglecting important aspects such as loss function analysis and other performance indicators. This gap in research presents an opportunity to leverage the power of neural networks for more comprehensive analysis of the dataset. By incorporating neural network implementations, future studies can delve deeper into model performance, understanding not only accuracy but also examining loss metrics, model convergence, and potential improvements in predictive capabilities. This approach would enhance the understanding of the dataset and provide a more nuanced evaluation of model effectiveness compared to traditional machine learning techniques.

IV. Approaches

Leveraging machine learning techniques, a classification model can be developed to accurately classify trips as either genuine or using fake GPS. Supervised learning algorithms like logistic regression, XGBoost, or neural networks can be trained on labeled data to discern these patterns and make predictions on unseen data. Regularization techniques and feature selection methods can further enhance the model's performance and interpretability. By iteratively refining the model with feedback from its predictions, it can effectively distinguish between genuine and fake GPS usage, aiding in the detection and prevention of fraudulent activities.

V. Methodology

A. Data Preparation and Preprocessing

Data preparation involved addressing missing values in the dataset, primarily in the *altitude_in_meters* column, which exhibited a significant presence of NA values. Given its limited relevance to the spatial analysis and the overwhelming number of missing values, it was deemed appropriate to drop the column. This decision ensured the integrity of subsequent analyses by focusing solely on essential spatial coordinates, namely latitude and longitude. Furthermore, preprocessing steps involved standardizing the format of temporal data, leveraging the linux_date column for enhanced precision, and removing redundant date and hour columns to streamline the dataset for further analysis.

B. Exploratory Data Analysis (EDA)

The exploratory data analysis (EDA) phase provided critical insights into the characteristics and distribution of the dataset. Visualizations were instrumental in understanding the prevalence of spoofed and non-spoofed rides across different service types, uncovering potential data imbalances that may impact model training and performance. Additionally, the analysis revealed intriguing patterns in GPS accuracy across service types, with implications for the occurrence of fake GPS signals and overall data quality. These findings underscored the importance of comprehensive exploratory analyses in uncovering trends and anomalies within the dataset. Summary of findings can be found at Appendix: Additional Comments On Exploratory Data Analysis.

C. Feature Engineering

Feature engineering played a pivotal role in enhancing the dataset with meaningful variables that capture various aspects of driver behavior and order dynamics. One-hot encoding of categorical columns ensured the representation of categorical variables in a format conducive to machine learning algorithms, mitigating issues related to ordinality and hierarchy. Furthermore, the derivation of temporal and spatial metrics, such as time differences between pings, distances traveled, and driver speeds, enriched the dataset with actionable insights into the sequential and spatial aspects of driver behavior. These engineered features laid the foundation for more robust and accurate predictive modeling. Summary of features engineered can be found at Appendix: Additional Comments On Feature Engineering.

D. Train-Test Split

In the process of splitting the dataset into training and testing sets based on order IDs, a meticulous approach was adopted to maintain evaluation integrity and prevent data leakage. Each order ID was meticulously assigned to either the training or testing set, ensuring that the model solely learns from orders within its designated set, thereby averting any inadvertent influence of testing data on model training that could inflate performance scores. Leveraging the train_test_split function with the stratify parameter set to the labels ensured consistent label distribution (0 and 1) across both sets, enhancing the reliability of evaluation metrics. Allocating 25% of order IDs to the testing set and utilizing a fixed random seed (4266) for reproducibility further reinforced the consistency of the split. This methodological rigor ultimately bolstered the reliability of assessing the model's generalization to unseen data, yielding more precise performance estimates and enabling equitable comparisons between different models.

1) For classical models: Subsequently, after the split, the resulting datasets, train_data and test_data, underwent feature and target variable construction. From train_data, the feature matrix X_train was derived by excluding the 'label' column, while the target vector y_train was obtained by selecting only the 'label' column. Similarly, from test_data, X_test was formed by dropping the 'label' column, and y_test was isolated from the 'label' column. This systematic structuring ensured appropriate model training and evaluation inputs. Following this step, normalization using the StandardScaler from the sklearn package was applied to uniformly scale features, enhancing and stabilizing model performance.

2) For neural network models: In addition to train and test data, we will need validation data as well to train the neural networks. This is done by splitting train_data again into train_data and validation_data. This time, we allocated 15% of the original train_data for validation_data utilizing the same fixed random seed (4266). Furthermore, given the nature of neural networks, the input data requires a fixed dimension tensor for the model to train on. Thus, we need to define certain dimensions of the tensor that we want. Firstly, we will need to limit the number of sequential pings to a certain threshold, 600 (as most rides have less than 600 pings), by padding shorter rides with zeros while truncating longer rides to contain only 600 pings. Thus, the input tensor dimensions will be $N \times M \times C$, whereby:

$$N = \text{ number of rides}$$

$$M = \text{ number of sequential pings}$$

$$C = \text{ number of features}$$

Then, the data are scaled separately using min-max scaling. Using min-max scaling is ideal for our dataset because it keeps the minimum value at 0, which is important for maintaining certain data meanings without affecting padding. Our data mostly consists of non-negative values, except for bearings, which can be easily handled by taking their absolute values before scaling.

E. Models

1) Logistic Regression: In the initial phase of our model testing, we employed Logistic Regression, a statistical model that is notably esteemed for its simplicity and efficacy in binary classification problems. This method was specifically chosen for its interpretability and the linear relationship it assumes between the independent variables and the logarithm of the odds ratio. To ensure a rigorous evaluation of our model's performance, we implemented stratified k-fold cross validation. This technique partitioned the data into five distinct folds, preserving the proportionate representation of each class across folds, thereby maintaining the integrity of the dataset's class distribution. Utilising Logistic Regression allowed us to establish a foundational understanding of the dataset's dynamics under standard classification conditions.

Thereafter, we introduced regularisation to the Logistic Regression algorithm to enhance its prediction accuracy and robustness. Regularisation technique such as L2 (Ridge) was-applied, aimed at preventing overfitting and improving model generalisation across unseen data. This was critical as it not only helped manage the multicollinearity among the features but also ensured a more stable model performance across different subsets of data, which was crucial for achieving consistent and reliable predictions.

In the final enhancement to the modelling process, we integrated the Synthetic Minority Over-sampling Technique (SMOTE). This approach was crucial in addressing the significant class imbalance present in our dataset. By artificially generating samples for the minority class, SMOTE facilitated a more balanced class distribution, crucial for reducing bias during the model's training. The combination of Logistic Regression with SMOTE allowed for a more balanced and equitable model, which significantly improved our classification performance on imbalanced datasets.

2) XGBoost Classifier: In this iteration, we leveraged the formidable capabilities of XGBoost. To ensure rigorous evaluation, we employed stratified k-fold cross-validation, dividing the data into five folds while preserving class distribution integrity. Additionally, we addressed class imbalance by integrating the SMOTE technique, augmenting the minority class with synthetic samples. This fusion of XGBoost with cross-validation and SMOTE presented a potent approach adept at tackling skewed datasets and delivering sharper predictions.

In a subsequent enhancement, we introduced parameter tuning using RandomizedSearchCV to further elevate classifier performance, particularly crucial for imbalanced datasets. This systematic exploration of hyperparameters aimed to pinpoint the optimal configuration, refining model generalization and achieving superior classification outcomes. Through RandomizedSearchCV, we fine-tuned key XGBoost hyperparameters while addressing regularization and loss reduction to mitigate overfitting. This endeavor aimed to uncover the optimal configuration that maximized performance metrics, yielding a discerning classifier tailored to the dataset at hand.

Following the fitting of RandomizedSearchCV, the finest estimator underwent further training on oversampled data engendered by SMOTE. This step ensured the model learned from a balanced representation of both classes, enhancing its ability to distinguish between them during prediction. By amalgamating SMOTE with parameter tuning, our approach aimed to craft a robust XGBoost classifier adept at grappling with class imbalance and furnishing enhanced classification performance.

3) Temporal Convolutional Neural Network: Temporal Convolutional Neural Networks (TCNNs) are a type of neural network designed to process sequential data, such as time series like our problem statement or text, by applying convolutional operations over time.

In simpler terms, TCNNs work by scanning through a sequence of data, like speed over time representing time steps, and learning patterns or features from small windows of consecutive data points, capturing things like trends, cycles, or important events in the sequence.

The main advantage that TCNN has to offer is because it can inherently parallelize computations across time steps due to the structure of convolutional operations. This leads to faster training and inference times compared to RNNs, which process data sequentially. Hence, needing less computation power to train the model.

Initially, we embarked on developing our own Temporal Convolutional Neural Network (TCNN) model from scratch. However, in exploring for more solutions for our dataset, we discovered a pre-built package for Temporal Convolutional Networks (TCNs) [2]. We also conducted experiments with these packages and found them to be highly effective for our requirements due to their intuitive and straightforward implementation, especially in areas such as hyperparameter tuning.

4) Long Short-Term Memory: Similar to TCNN, we also explored the use of RNN, especially LSTM, to handle this dataset. RNN would be appropriate to handle this sequence classification problem as it is designed to process sequential data. Specifically, we chose the LSTM architecture for our problem.

LSTM networks are particularly effective for capturing long-term dependencies and patterns in sequential data, compared to normal RNN architecture. In our problem, there may exist long-term dependencies, such as checking if a driver's latest location matches with their route history, which traditional feedforward networks struggle to capture. However, with the presence of forget, input, and output gates, LSTMs are better able to remember past information, which is important in identifying routes with anomalies at certain timesteps. Thus, LSTMs, with their ability to retain information over long sequences, are likely well-suited for this task.

We aim to conduct a comprehensive set of hyperparameter tuning to understand the impact of various architectural elements on model performance. Our experiments will focus on manipulating layers such as batch and layer normalization, masking, dropout, L1/L2 regularization, and varying the configuration of LSTM layers (single, stacked, bidirectional) to discern their effects on accuracy and BCE loss.

5) AutoEncoders: Another approach that we explored was the use of Autoencoders. Particularly, the typical Autoencoder (AE) and Variational Autoencoders (VAEs). Autoencoders are ideal for anomaly detection as they are great in learning efficient representations of data and can hence highlight deviations from learnt patterns.

In our data, a labelled fake trip is labelled as so for all GPS ping signals. Whether a particular GPS signal in the sequence, or the entire sequence was faked, we don't know.

Our hypothesis is that faked GPS signals would have a higher variation in error or produce artifacts in the GPS signal data if there was a mix of legit and non-legit GPS signals. (Eg. A driver at the start of the trip fakes his GPS to a customer hotspot, and once the ride is accepted, he turns off the fake GPS signal and it jumps back to his original location).

The Autoencoders can then identify these deviations in the sequence and reflect these fake trips accordingly.

More information of the model training can be found at Appendix:Additional Notes on Model Training

VI. Results and Evaluation

In evaluating the performance of our model for binary classification of GPS signal authenticity, a comprehensive set of metrics was employed to provide a holistic assessment of its predictive capabilities. These metrics include accuracy, precision, recall, F1 score, Binary Cross Entropy Loss, and ROC-AUC. Accuracy measures overall correctness, precision emphasizes avoiding false positives, recall focuses on capturing all positive instances, and the F1 score balances precision and recall.

In this project, prioritizing recall is paramount to minimize misclassification of genuine signals as spoofed, crucial for user convenience and security. The confusion matrix provides deeper insights into classification results, detailing true positives, true negatives, false positives, and false negatives. Additionally, the ROC-AUC metric accounts for the temporal nature of data, assessing the model's discriminative ability across various decision thresholds.

Furthermore, the choice of the Binary Cross Entropy loss function guides the model's learning process by quantifying the disparity between predicted probabilities and actual binary labels, particularly crucial in scenarios with significant consequences for misclassification. By leveraging these diverse metrics and methodologies, a nuanced understanding of the model's performance is obtained, facilitating informed decision-making and optimization for real-world applications in GPS signal authenticity classification.

Our evaluation metrics on the test dataset and the corresponding model results are as seen in Table I.

TABLE I
Model Performance on Test Data Summary

	LogReg	XGBoost	TCNN	LSTM	AE
Error	0.56	5.73	0.46	0.40	2.04*
Accuracy	71.4	84.1	77.5	81.8	70.0
Precision	75.3	92.2	84.3	89.4	100.0
Recall	88.3	84.5	83.4	84.1	0.0

* Means the Error is Reconstruction Error instead of BCE. Whereby Reconstruction error (RE) is the metric to evaluate the deviation of the reconstructed data from the actual data.

A. Logistic Regression

The comprehensive evaluation of the Logistic Regression classifier across various setups provided valuable insights into its performance metrics. Initially implemented without sampling, the model demonstrated a solid accuracy of 72.11%, with a precision of approximately 76.16% for the positive class and a recall of 87.80%. These metrics indicated the model's proficiency in minimizing false positives and effectively capturing true positives. The balanced F1-score of 82% and an Area Under the Curve (AUC) of 0.614 further reflected its reasonable discriminative power. Upon integrating L2 Regularization, there was a slight decrease in overall accuracy to 71.43%. However, the model exhibited improved precision (75.31%) and an impressive AUC of 0.789, suggesting enhanced generalization capabilities and a better ability to distinguish between class labels. The recall remained high at 88.29%, maintaining robust sensitivity to genuine positives. The addition of SMOTE notably improved the accuracy to 75.09%, the highest observed in our tests, and significantly increased precision to 92.14%, crucial for reducing false positive rates. This setup, however, showed a reduction in recall to 70.57%, reflecting the challenges of balancing the effects of synthetic sample generation. The consistent F1-score and AUC values (80% and 0.782, respectively) indicated that the model maintained a strong performance, especially in precision-oriented scenarios. Overall, each enhancement to the Logistic Regression model—whether through Regularization or SMOTE—brought distinct improvements. Regularization primarily boosted the model's ability to differentiate between classes without sacrificing sensitivity, while SMOTE enhanced precision significantly, albeit at some expense to recall.

B. XGBoost Classifier

The evaluation of the XGBoost classifier for GPS signal authenticity classification yielded promising results across various performance metrics. Without sampling, the model achieved an accuracy of 83.77%, with strong precision (0.89) and recall (0.88) for the positive class, indicating its proficiency in minimizing false positives and capturing genuine signals effectively. With the addition of SMOTE, the model's accuracy improved to 84.11%, showcasing robust capability in classifying data accurately, while achieving a higher precision of 0.92. Furthermore, the recall remained high at 0.85, emphasizing the model's effectiveness in identifying actual positive data points. Integration of SMOTE and parameter tuning did not significantly enhance performance, with metrics showing similar results to SMOTE alone. However, comparative evaluation revealed distinct advantages of using SMOTE, showcasing superior metrics including accuracy, precision, recall, and AUC, validating its efficacy in addressing class imbalance and improving classification performance.

C. Temporal CNN

The evaluation of the Temporal CNN classifier on the GPS signal authenticity classification task demonstrates promising results across multiple performance metrics, although less effective than when using classical models. With an overall accuracy of 77.50%, the model exhibits some proficiency in correctly classifying a significant portion of the data points.

TABLE II
TCNN PERFORMANCE ON TEST DATA SUMMARY

	Own Architecture		Keras-TCN
	Base	Oversampling	
BCE	0.46	0.57	0.50
Accuracy	77.5	75.2	74.3
Precision	84.3	76.4	80.4
Recall	83.4	76.5	83.7

Additionally, the precision for the positive class is 84.3%, indicating a strong capability to minimize false positive predictions, essential for scenarios where misclassified genuine signals could pose substantial risks.

Our custom Temporal CNN (TCNN) architecture achieved superior performance(Table II) across most evaluation metrics compared to the Keras-TCN implementation. While recall was slightly lower (0.4% difference), the custom model excelled in other areas. Notably, naive oversampling of the spoofed GPS data, a minority class, significantly degraded performance, particularly in terms of loss. Therefore, we opted for our original TCNN architecture as the final model.

Furthermore, the Area Under the Curve of the Receiver Operating Characteristic (AUC ROC) for the Temporal CNN classifier is measured at 0.82, a robust indicator of its discriminatory power. This value is on par with classical models, affirming the model's effectiveness in distinguishing between genuine and spurious GPS signals.

In terms of loss, the Binary Cross-Entropy (BCE) loss stands at 0.463, significantly lower compared to traditional models commonly used in this context. This lower loss underscores the model's ability to more effectively refine predictions during training, resulting in improved performance and generalization on unseen data as compared to classical models.

D. LSTM

TABLE III
LSTM PERFORMANCE ON TEST DATA SUMMARY

	1-Layer		2-Layer	Bidirectional
	Base	Oversampling		
BCE	0.40	0.45	0.42	0.42
Accuracy	81.8	78.3	79.9	79.8
Precision	89.4	88.4	90.0	92.0
Recall	84.1	79.5	80.3	78.0

Across different LSTM architectures, single-layer LSTM model (base) performs relatively well at 81.8%, compared to double-layer (79.9%) and bidirectional LSTM (79.8%), showing that more complex models tend to overfit the training data, making it difficult to generalize to test dataset. Comparing base single-layer model with one trained on oversampled data, the base layer trained on the original data performed significantly much better than the oversampled one in terms of accuracy, loss, and recall (Table III).

Furthermore, comparing LSTM to other classical approaches we have discussed, it also performed relatively well compared to logistic regression and TCNN, showing that the time series data contains useful information about fake/genuine trip classification. Nevertheless, it still performed marginally worse than XGBoost, showing that summary statistics might be more appropriate in this setting.

In terms of recall, base single-layer model performed comparably (84.1%) to other neural network and classical approaches. However, the LSTM base model has the highest AUC ROC of 0.87 as well as lowest BCE loss of 0.400, showing that the predicted probabilities match our target response most closely compared to other approaches.

E. Autoencoders (AE)

The Autoencoder (AE) and Variational Autoencoder (VAE) models were employed to identify anomalous trips based on GPS data reconstruction error. In this context, anomalies are those with high reconstruction errors, indicative of potentially fraudulent trips. The performance metrics on the test data are summarized in Table IV

TABLE IV
AE PERFORMANCE ON TEST DATA SUMMARY

	AE	VAE
BCE	2.04	4.73
Accuracy	70.0	70.0
Precision	100.0	0.00
Recall	0.005	0.0

The AE achieved perfect precision at 100%, which indicates its strong ability to avoid misclassified false positives. This is essential, as wrongly assuming a genuine customer/driver of fraud has negative impacts on customer trust and the company's reputation.

The high accuracy rate of 70.0% for both models misrepresents the models' ability to identify fraudulent trips, and is an artifact of the dataset's unbalanced nature.

Its recall is virtually **zero**, indicating that while it is highly confident in its' detected cases, it misses almost all actual fraudulent instances. Given the AEs ability to identify anomalies well, this could hint that the trips between genuine and faked trips are similar, and hence there isn't a stark difference in reconstruction loss. A further analysis is conducted into this in Neural Network Models, confirming our hypothesis to be *false*, showing that the distribution of reconstruction errors between fraudulent and genuine trips to be similar.

Additional information of overall model evaluation can also be found at Appendix: Additional Notes on Model Evaluation.

VII. ERROR ANALYSIS

Error analysis in neural networks involves employing various methods to understand and address model performance and shortcomings. Here are some that we have employed.

A. Confusion Matrix

One fundamental approach is the Confusion Matrix Analysis, which visually represents the distribution of predicted versus true classes, particularly valuable for binary classification tasks. By scrutinizing this matrix, we can pinpoint which classes are frequently misclassified and identify areas of model proficiency.

B. Loss

In addition to serving as a tool for result analysis, Binary Cross Entropy (BCE) loss also plays a critical role in error analysis as a diagnostic tool, indicating whether the model is overfitting or underfitting, thereby enabling parameter adjustments for optimization.

C. Model-Specific Error Analysis Approaches

Given the distinct approaches to data handling for classical versus neural network models, we will also be employing unique methodologies of error analysis for each case. Below, we outline our specific methods for each classification of models.

1) Classical Models: To understand the discrepancies between predicted labels and ground truth labels, we performed error analysis. Leveraging methods such as Cosine Similarity, SHAP (SHapley Additive exPlanations), and feature importance plots, we gained valuable insights into the factors influencing model predictions. Cosine similarity helped identify rows with similar features but different labels. Both logistic regression and XGBoost classifier have identified 'accuracy_in_metres' and 'unavailable_count' as most important features in driving the classification performance.

2) Neural Network Models: As neural network architectures are significantly more difficult to interpret than classical approaches, we propose two methods to analyse errors: feature importance and plotting misclassified trips.

Firstly, for feature importance, we average the absolute weights going into the LSTM layer for each feature, and then scale accordingly to take into account the different scales for each feature. The final value for each feature signifies feature importance, as higher average absolute weight for a particular feature means the model takes into account the feature more compared to others. We found that *driver_status* was the most important feature in classifying fake/genuine trips.

Second, we plotted misclassified trips to gain insights into what causes the model to classify specific observations wrongly. We discovered that some fake trips are surprisingly similar to genuine trips in terms of accuracy and trajectory, leading the model to wrongly predict them as genuine. Lastly, because the model does not possess knowledge of the road connections, it cannot capture the feasibility between coordinate changes. For example, some fake trips contain coordinate jumps that may seem small and feasible, but in actually are unrealistic. This includes moving across unconnected roads, or unrealistic large distance jumps. connections.

Additional information of error analysis done can be found at Appendix: Additional Notes on Error Analysis.

VIII. Discussion and Conclusion

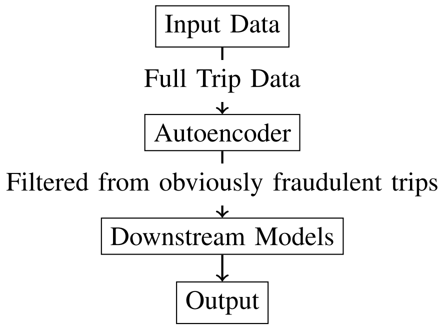
Based on the findings, our group has effectively developed and executed a methodology tailored to classify fake GPS signals within the Gojek application's datasets up to 84.1% accuracy, focusing on PING behavior with the classification task leveraged on LSTM as the chosen machine learning algorithm.

Despite XGBoost demonstrating higher accuracy in the classification task, the LSTM model exhibited significantly lower loss, which ultimately led to the selection of LSTM for this particular application. The lower loss observed with LSTM suggests improved refinement in predictions during training, potentially leading to enhanced generalization and performance on unseen data. Therefore, the decision to favor LSTM over XGBoost was driven by the importance of optimizing loss metrics for this specific GPS signal authenticity classification task within the Gojek application's datasets.

IX. Future Works

Moving forward, there are several promising avenues for research and development in this area. One direction could involve refining the LSTM algorithm to enhance classification outcomes by using knowledge of road connections from maps to tackle the issue of coordinate jumps in Neural Network Models. Another improvement could possibly be by exploring advanced architectures like Transformers with suitable computational tools. Additionally, the dataset created in this study offers a valuable opportunity for comparative analysis with alternative machine learning algorithms. This comprehensive evaluation could yield valuable insights for advancing detection methodologies within GPS datasets used by applications similar to Gojek such as Grab or even TADA. Exploring these directions could contribute to more effective and robust detection systems tailored to the unique challenges of cybersecurity risk in GPS signal authenticity classification.

Another additional potential avenue for improvement are building model ensembles. This allows us to leverage on strengths from different models, and make up for each others' weaknesses. For example, the strong precision of Autoencoders can be used as a first filter to remove all obviously anomalous trips, before sending data for further inference in downstream models.



Sequential processing of data: Input first goes through an Autoencoder which filters obviously anomalous trips, before trickling to downstream models.

Fig. 1. Flowchart of the Sequential Ensemble Model

References

[1] Gibran Erlangga. (2019). Fake GPS Detection. Kaggle. https://kaggle.com/competitions/dsbootcamp10

[2] Remy, P. (2020). Temporal Convolutional Networks for Keras [GitHub repository]. GitHub. https://github.com/philipperemy/keras-tcn

[3] Kingma, D. P., & Ba, J. (2014, December 22). Adam: A Method for Stochastic Optimization. ArXiv.org. https://arxiv.org/abs/1412.6980

[4] Dozat, T. (n.d.). Workshop track -ICLR 2016 INCORPORATING NESTEROV MOMENTUM INTO ADAM. https://openreview.net/pdf/OM0jvwB8jIp57ZJjtNEZ.pdf

[5] Ioffe, S., & Szegedy, C. (2015, March 2). Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv.org. https://arxiv.org/abs/1502.03167

[6] Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016, July 21). Layer normalization. arXiv.org. https://arxiv.org/abs/1607.06450

[7] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (1970, January 1). Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research. https://jmlr.org/papers/v15/srivastava14a.html

X. Appendix

A. Tables

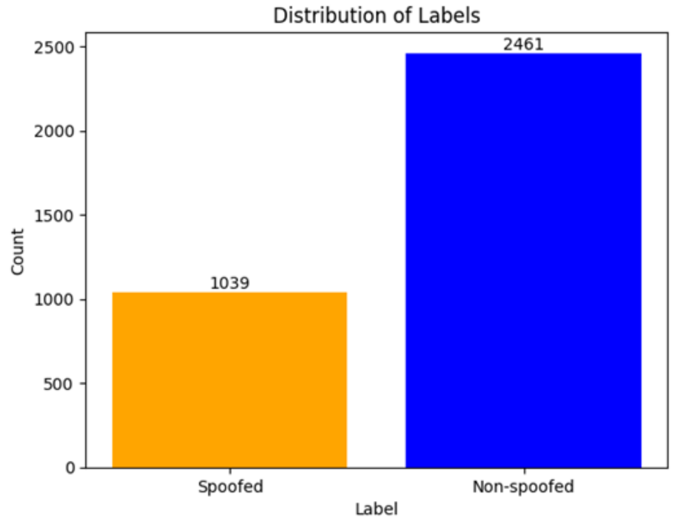
TABLE V
GOJEK Data attributes and value

Attributes	**Value**	**Type**
order_id	unique id of each ping	text
service_type	GoFood or GoRide	category
driver_status	unavailable, available, otw_pickup, otw_dropoff	category
date	2018-02-05 until 2018-03-23	date
hour	time in hours	numeric
seconds	time in seconds	numeric
latitude	latitude coordinate	numeric
longitude	longitude coordinate	numeric
altitude_in_meters	altitude coordinate in meters	numeric
accuracy_in_meters	location accuracy in meters	numeric
label	0=fake, 1=true	binary

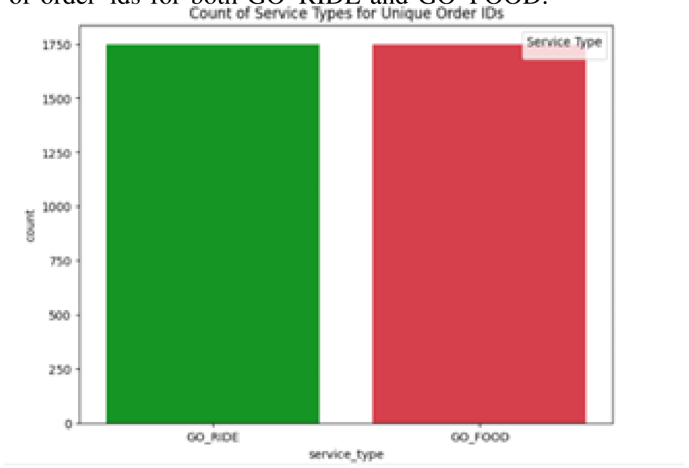
B. Additional Comments On Exploratory Data Analysis

This section will show some of the plots done to understand the distribution of data across the different service types and labels. We prioritize examining the order_id, which uniquely identifies each ride, as well as the label as a foundational basis for assessing the balance of other features. This is particularly significant because each order_id is linked with the label to indicate spoofed or non-spoofed rides, offering valuable insights into the distribution and characteristics of the dataset.

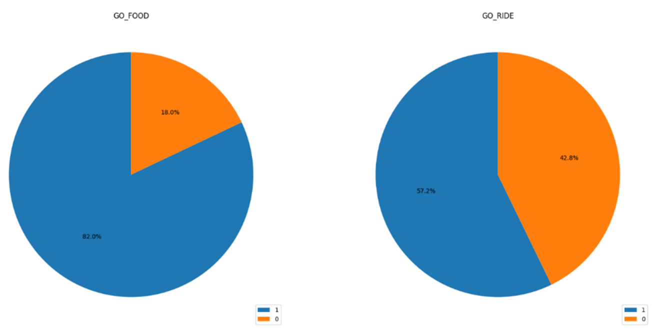
1) order_id: Among the 3500 unique orders in the dataset, approximately 33% of the provided data is identified as spoofed. This highlights an inherent data imbalance, suggesting the potential need for further balancing during model training if the models exhibit performance issues.



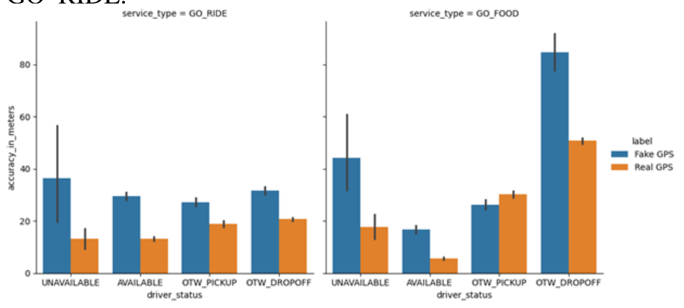
2) service_type: The dataset contains an equal distribution of order_ids for both GO_RIDE and GO_FOOD.



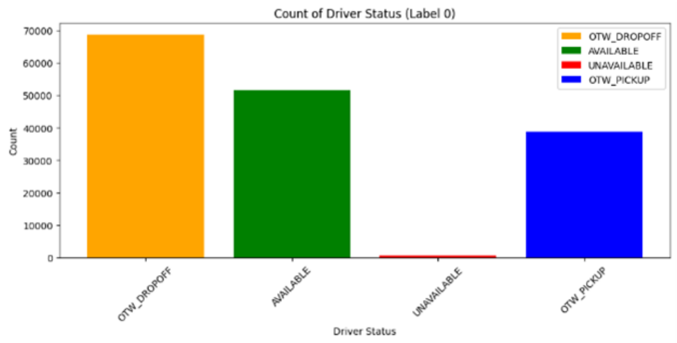
However, it is noted that GO_RIDE has a larger proportion of Fake GPS signals as compared to GO_FOOD.

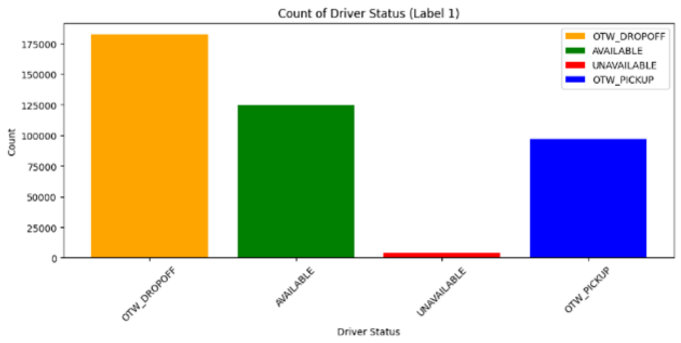


3) accuracy_in_metres: The plot illustrates the distribution of accuracy_in_metres across different driver_status categories for each service_type. Notably, the bars representing accuracy_in_metres tend to be higher for GO_FOOD than for GO_RIDE. This discrepancy is evident in the mean accuracy values: for GO_RIDE, the mean accuracy of Fake GPS dips as low as 40 meters, whereas the accuracy of Real GPS reaches levels as precise as 10 meters. Lower values on the plot signify higher accuracy, implying that a reading as low as 10 meters indicates an exceptionally accurate location detection capability. This distinction underscores the disparity in location accuracy between the two service types, with GO_FOOD generally demonstrating a wider range of accuracy and a higher likelihood of Fake GPS occurrences compared to GO RIDE.



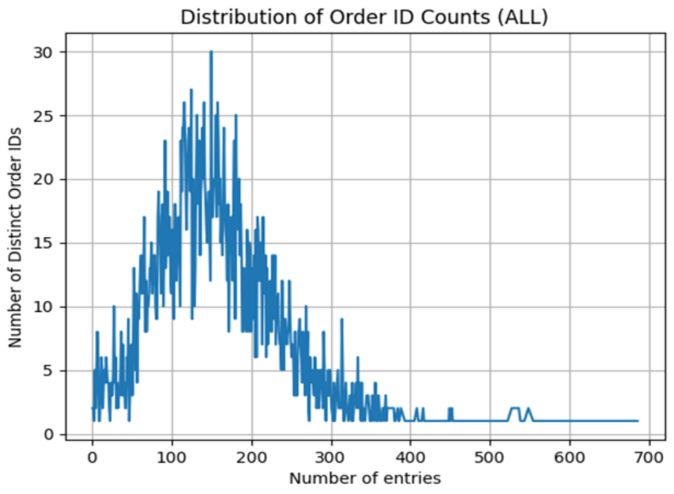
The plot illustrates the distribution of accuracy_in_metres across different driver_status categories for each service_type. Notably, the bars representing accuracy_in_metres tend to be higher for GO_FOOD than for GO_RIDE. This discrepancy is evident in the mean accuracy values: for GO_RIDE, the mean accuracy of Fake GPS dips as low as 40 meters, whereas the accuracy of Real GPS reaches levels as precise as 10 meters. Lower values on the plot signify higher accuracy, implying that a reading as low as 10 meters indicates an exceptionally accurate location detection capability. This distinction underscores the disparity in location accuracy between the two service types, with GO_FOOD generally demonstrating a wider range of accuracy and a higher likelihood of Fake GPS occurrences compared to GO_RIDE.





This plot shows the distribution of driver status for each of the labels. It seems that the data is distributed in a similar way for both labels with driver status = UNAVAILABLE being slightly larger in label 1.

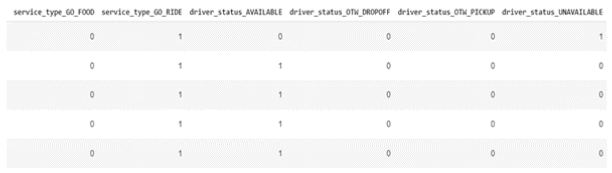
4) Number of pings per order_id: Based on an analysis, it appears that the distribution of the number of pings per order_id remains relatively consistent between spoofed and non-spoofed rides, with the majority having fewer than 300 pings.



C. Additional Comments On Feature Engineering

1) Categorical Features: We first performed one-hot encoding on categorical columns, essentially creating binary columns for each category present in the original categorical variable.

a. In this case, for the service_type variable containing ”GO_RIDE” and ”GO_FOOD”, two new binary columns would be created (Refer to Table below).



service_type_GO_FOOD	service_type_GO_RIDE	driver_status_AVAILABLE	driver_status_OTW_DROPOFF	driver_status_OTW_PICKUP	driver_status_UNAVAILABLE
0	1	0	0	0	1
0	1	1	0	0	0
0	1	1	0	0	0
0	1	1	0	0	0
0	1	1	0	0	0

b. Similarly, for the driver_status variable containing ”AVAILABLE”, ”UNAVAILABLE”, ”OTW_PICKUP”, and ”OTW_DROPOFF”, four new binary columns would be created, each representing one of these statuses (Refer to Table above).

This approach of one-hot encoding is chosen over label encoding primarily to avoid introducing unnecessary ordinality or hierarchy in the categorical variables. Label encoding assigns integer values to each category, thereby implying an order or hierarchy among them, which might not be present or meaningful in the original data. For instance, if we were to label encode the service_type variable with ”GO_RIDE” as 0 and ”GO_FOOD” as 1, it might inadvertently imply that ”GO_FOOD” trips are somehow more important than ”GO_RIDE” trips, which is not necessarily the case.

2) Numerical Features: This section will cover the feature engineering done on the numerical columns

a. We first converted seconds column to linux_date. However, we noticed that the existing date and hour column did not match with the converted linux_date column. Given the precision offered by the linux_date column, we opted to utilize it exclusively and subsequently removed the hour and date columns. We concluded that the discrepancy in the date and hour columns are likely due to a data mismatch or errata in the dataset provided.
b. Counting the occurrences of ’UNAVAILABLE’ driver_status for each order provided insight into the availability of drivers during the order process. This was named as unavailable_count.
c. The dataset was then sorted based on the order_id and seconds to facilitate sequential analysis of events related to each order.
d. Additionally, time differences between consecutive pings were calculated to understand the temporal aspect of events. This was named as time_diff_seconds. The distances between consecutive pings were derived using the Haversine formula, offering spatial context to the movement of drivers during orders. This was called as distance.
e. Speed calculations were performed by dividing distance by time difference, providing information about the speed of drivers labelled as speed. Moreover, latitude and longitude differences, as well as their rates of change per second, were computed to capture directional changes and movement patterns. They were stored as lat_diff, lon_diff, lat_rate and lon_rate.
f. Additionally, bearing was calculated to determine the direction of driver from previous timestep to the next one:

$$\text{bearing} = \arctan\left(\frac{\Delta y}{\Delta x}\right).$$

g. Geospatial aggregates such as mean, median, min, max, and count were computed for latitude and longitude coordinates, providing summary statistics about the spatial distribution of drivers during orders. Similar aggregates were calculated for speed, distance, accuracy, time differences, latitude rate, longitude rate, and OTW_pickup times, offering comprehensive insights into various aspects of driver behavior and order dynamics.

Overall, these feature engineering steps aimed to enhance the dataset with meaningful variables that capture different facets of driver behavior and order characteristics, facilitating deeper insights and more accurate predictions in subsequent analyses.

D. Additional Notes on Model Training

1) Logistic Regression: In our study employing Logistic Regression with Regularisation, we decided to implement L2 regularisation (Ridge). Here is a detailed discussion of why we chose L2 over L1 (Lasso):

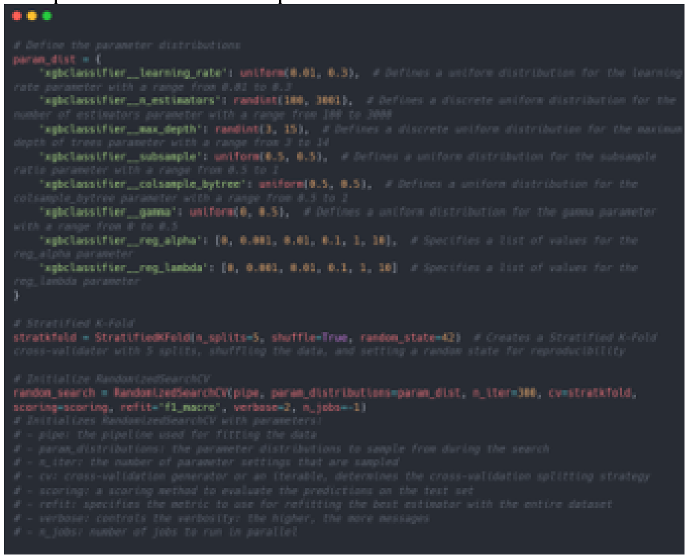
a. Coefficient Shrinkage and Model Stability: L2 regularization applies a penalty equal to the square of the magnitude of coefficients. This approach, unlike L1 which might zero out coefficients entirely, gently shrinks the coefficients but keeps all features in the model. This characteristic was particularly crucial for our analysis as it prevents any single feature from exerting too much influence on the predictions. I believed that maintaining all features would be beneficial given our dataset’s complexity, where dismissing any variable could omit subtle but important information.

b. Control Overfitting: Another significant factor in favor of L2 was its capability to control overfitting. By penalizing the sum of the squares of the coefficients, L2 ensures that the model does not fit the noise in our training data too closely. This is particularly important in our scenario where the risk of overfitting is heightened by the relatively high dimensionality of our data. Reducing overfitting helps in enhancing the model’s generalization capabilities to new, unseen data sets, thereby improving predictive performance on external validation sets.

c. Flexibility and Simplicity in Implementation: L2 regularization is straightforward to implement using most standard statistical software, providing a practical advantage in terms of ease of use and computational efficiency. This ease of implementation, coupled with its effectiveness in various predictive modeling scenarios, made L2 an appealing choice for our logistic regression framework.

d. Consistent Impact Across All Coefficients: Unlike L1, which can be somewhat arbitrary in which coefficients it reduces to zero, L2 treats all coefficients in a more uniform manner. This uniform treatment aligns with our goal to build a balanced model where each feature is allowed to contribute, albeit modestly, to the predictive process. This approach seemed particularly appropriate for ensuring that our model remains comprehensive and inclusive of all potentially influential variables.

2) XGBoost Classifier: In our study employing XGBoost with the Synthetic Minority Over-sampling Technique (SMOTE), we meticulously performed hyperparameter tuning to optimize the model's performance.



```
# Define the parameter distributions
param_dist = {
    'xgbclassifier__learning_rate': uniform(0.01, 0.3),  # Defines a uniform distribution for the learning
rate parameter with a range from 0.01 to 0.3
    'xgbclassifier__n_estimators': randint(100, 3001),  # Defines a discrete uniform distribution for the
number of estimators parameter with a range from 100 to 3000
    'xgbclassifier__max_depth': randint(3, 15),  # Defines a discrete uniform distribution for the maximum
depth of trees parameter with a range from 3 to 14
    'xgbclassifier__subsample': uniform(0.5, 0.5),  # Defines a uniform distribution for the subsample
ratio parameter with a range from 0.5 to 1
    'xgbclassifier__colsample_bytree': uniform(0.5, 0.5),  # Defines a uniform distribution for the
colsample_bytree parameter with a range from 0.5 to 1
    'xgbclassifier__gamma': uniform(0, 0.5),  # Defines a uniform distribution for the gamma parameter
with a range from 0 to 0.5
    'xgbclassifier__reg_alpha': [0, 0.001, 0.01, 0.1, 1, 10],  # Specifies a list of values for the
reg_alpha parameter
    'xgbclassifier__reg_lambda': [0, 0.001, 0.01, 0.1, 1, 10]  # Specifies a list of values for the
reg_lambda parameter
}

# Stratified K-Fold
stratkfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)  # Creates a Stratified K-Fold
cross-validator with 5 splits, shuffling the data, and setting a random state for reproducibility

# Initialize RandomizedSearchCV
random_search = RandomizedSearchCV(pipe, param_distributions=param_dist, n_iter=300, cv=stratkfold,
scoring=scoring, refit="f1_macro", verbose=2, n_jobs=-1)
# Initializes RandomizedSearchCV with parameters:
# - pipe: the pipeline used for fitting the data
# - param_distributions: the parameter distributions to sample from during the search
# - n_iter: the number of parameter settings that are sampled
# - cv: cross-validation generator or an iterable, determines the cross-validation splitting strategy
# - scoring: a scoring method to evaluate the predictions on the test set
# - refit: specifies the metric to use for refitting the best estimator with the entire dataset
# - verbose: controls the verbosity: the higher, the more messages
# - n_jobs: number of jobs to run in parallel
```

a. Learning Rate: By widening the range for the learning rate, we anticipated improved convergence during training. This adjustment was expected to facilitate the discovery of more optimal solutions, potentially leading to faster convergence and enhanced model generalization.

b. Number of Estimators: The expansion of the range for the number of estimators was aimed at enabling the model to capture more complex patterns in the data. Consequently, we expected an enhancement in the model's overall performance due to its increased capacity to adapt to intricate data structures.

c. Maximum Depth of Trees: By extending the range for the maximum depth of trees, we anticipated the model's ability to capture deeper and more intricate relationships within the data. However, careful monitoring was necessary to prevent overfitting, as deeper trees could potentially lead to an increased risk of overfitting.

d. Subsample Ratio: Adjusting the range for the subsample ratio aimed to introduce variability in the training process, potentially improving the model's generalization capabilities. We expected this variation to promote diversity among the trees in the ensemble, leading to more robust and adaptable models.

e. Colsample by Tree: The adjustment of the colsample by tree parameter was expected to mitigate overfitting tendencies by controlling the number of features considered for splitting at each node. By promoting diversity among the features used for tree construction, we anticipated improved model generalization and performance on unseen data.

f. Gamma: Fine-tuning the gamma parameter aimed to strike a balance between model complexity and regularization. We expected this adjustment to help prevent overfitting by controlling the minimum loss reduction required for further partitioning on a leaf node, thereby promoting simpler and more interpretable models.

g. Regularization Parameters (Alpha and Lambda): By exploring different levels of regularization through the adjustment of alpha and lambda, we expected to mitigate overfitting tendencies and enhance the model's generalization capabilities. Fine-tuning these parameters aimed to strike an optimal balance between model complexity and regularization, ultimately leading to improved performance on unseen data.

In our pursuit of hyperparameter optimization, we opted for RandomizedSearchCV over GridSearchCV. This decision was driven by the vast search space of hyperparameters and the desire for efficient exploration. RandomizedSearchCV offers the advantage of sampling a specified number of candidates from a parameter space distribution, providing a more computationally feasibl e approach.

The utilization of RandomizedSearchCV allowed us to efficiently explore a diverse range of hyperparameter combinations without exhaustively searching through every possible combination, thereby significantly reducing the computational burden. Despite its stochastic nature, RandomizedSearchCV provides a good balance between computational efficiency and the exploration of hyperparameter space.

During our experimentation, we observed that the time taken for each iteration of RandomizedSearchCV averaged around 30 minutes, with a total fitting time of approximately 30 minutes for 300 candidates across 5-fold cross-validation. This time investment remained consistent across multiple iterations, reaffirming the efficiency of RandomizedSearchCV in comparison to the more exhaustive GridSearchCV which was still running despite hours, likely due to our computational capability constraints.

Considering the computational demands and time efficiency observed with RandomizedSearchCV, it became evident that GridSearchCV would likely require significantly more time to explore the same hyperparameter space exhaustively. Hence, the adoption of RandomizedSearchCV proved to be a pragmatic choice for efficiently tuning hyperparameters and optimizing the XGBoost classifier with SMOTE integration.

3) Temporal CNN: Our experiments involve varying key parameters of the Temporal CNN model to optimize its performance in terms of accuracy and loss metrics. The following parameters were systematically adjusted and evaluated:

a. Number of Layers: We experimented with different depths of the CNN architecture, ranging from shallow to deep networks, to understand how the number of layers affects model performance.
b. Activation Functions: Two primary activation functions were compared: ReLU (Rectified Linear Unit) and Leaky ReLU. ReLU is a popular choice due to its simplicity and effectiveness, while Leaky ReLU helps mitigate the "dying ReLU" problem by allowing a small gradient for negative inputs. Eventhough our dataset would not contain much negative values, Leaky ReLU could still provide benefits in terms of training stability and learning robustness, especially in scenarios where the

dataset might contain noise or outliers that could produce negative values in the activation maps.

c. Optimizers: Various optimizers were tested for training the model, including SGD (Stochastic Gradient Descent), Adam (Adaptive Moment Estimation), Adamax [3], and Nadam [4]. However, we still found that using the base Adam optimizer provided the best result with minimal parameter tuning on the optimizer itself.

d. Regularization Techniques:

d.1. L2 Regularization: The inclusion of L2 regularization in the model's convolutional and dense layers helps to prevent overfitting by penalizing larger weights in the network. Thus allowing the model to recognize more generalizable patterns.

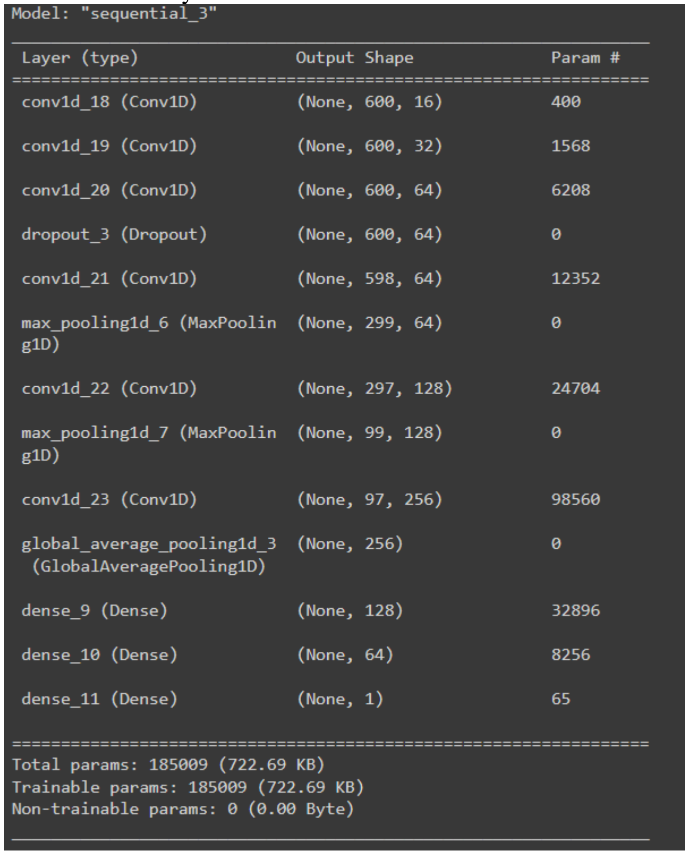
d.2. Dropout: By applying dropout between the convolution layers, where a fraction of neurons are randomly deactivated during training, the model becomes less sensitive to specific weights and neurons. This improves the network's ability to generalize to unseen data.

e. Optimization with Learning Rate Scheduling: The learning rate scheduler dynamically adjusts the learning rate during training, ensuring an optimal balance between convergence and optimization. Using a learning rate scheduler helps in stabilizing the training process by gradually reducing the learning rate over epochs. This approach can prevent the model from overshooting the minima and enable it to settle into a more optimal solution.

f. Max Pooling layers: Serves to downsample the feature maps produced by the Conv1D layers. By extracting the maximum activation within each pooling window, Max Pooling effectively reduces the spatial dimensionality of the feature maps. This reduction in spatial resolution helps in focusing on the most salient features learned by the preceding Conv1D layers, enhancing the model's ability to capture important temporal patterns in the GPS data. Our model seems to perform better when using max pooling, perhaps it is due to the noise that is being suppressed thus resulting in an improvement.

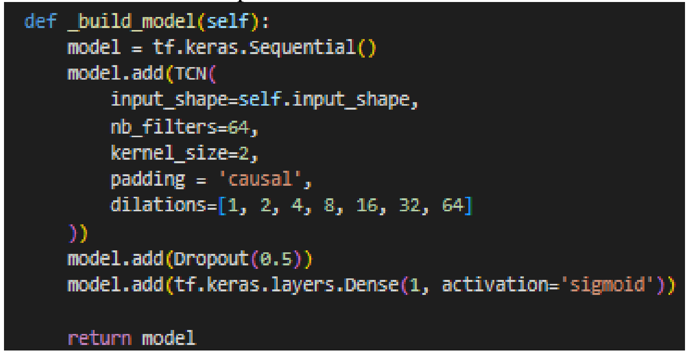
Overall, by meticulously tuning these components and techniques, we believe our Temporal CNN iteration represents a highly refined and effective approach tailored to the Gojek dataset and served our objectives.

The summary of the final TCNN model is as such:



```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_18 (Conv1D)          (None, 600, 16)           400

 conv1d_19 (Conv1D)          (None, 600, 32)           1568

 conv1d_20 (Conv1D)          (None, 600, 64)           6208

 dropout_3 (Dropout)         (None, 600, 64)           0

 conv1d_21 (Conv1D)          (None, 598, 64)           12352

 max_pooling1d_6 (MaxPoolin  (None, 299, 64)           0
 g1D)

 conv1d_22 (Conv1D)          (None, 297, 128)          24704

 max_pooling1d_7 (MaxPoolin  (None, 99, 128)           0
 g1D)

 conv1d_23 (Conv1D)          (None, 97, 256)           98560

 global_average_pooling1d_3  (None, 256)               0
  (GlobalAveragePooling1D)

 dense_9 (Dense)             (None, 128)               32896

 dense_10 (Dense)            (None, 64)                8256

 dense_11 (Dense)            (None, 1)                 65

=================================================================
Total params: 185009 (722.69 KB)
Trainable params: 185009 (722.69 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

We also experimented with pre-built TCNN packages such as keras-TCN [2] as mentioned above. Implementation was simple enough, the class TCN acts just like an LSTM layer with arguments to change the number of layers through nb_filters argument, kernel_size to tell the model how much of the previous sequence to depend on, and so on, making parameter tuning much easier. More information regarding the use of the package and its arguments can be found on the GitHub repository cited, but here is also a sample of how the model can be made:



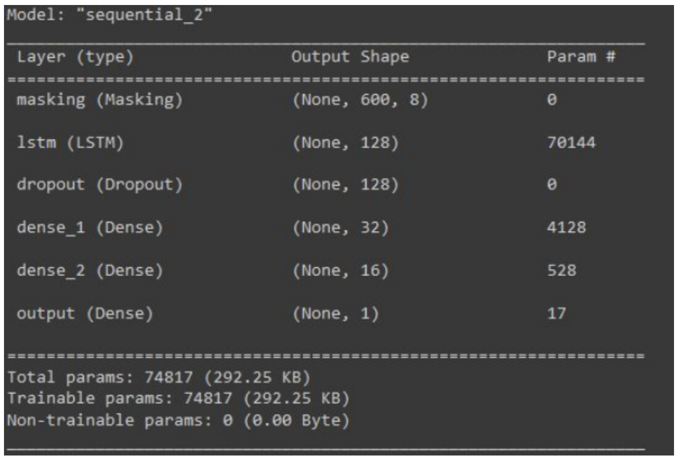
```
def _build_model(self):
    model = tf.keras.Sequential()
    model.add(TCN(
        input_shape=self.input_shape,
        nb_filters=64,
        kernel_size=2,
        padding = 'causal',
        dilations=[1, 2, 4, 8, 16, 32, 64]
    ))
    model.add(Dropout(0.5))
    model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

    return model
```

However, after experimenting with it, the results still fall short compared to those obtained with our custom-trained model. Consequently, we have opted to retain our originally designed model.

4) LSTM: We aim to conduct a comprehensive set of hyperparameter tuning to understand the impact of various architectural elements on model performance. Our experiments will focus on manipulating layers and hyperparameters as shown below:

a. Batch Normalization: Batch normalization has been proven effective in stabilizing and accelerating training by normalizing activations* within each batch [5]. We will systematically experiment with adding and removing batch normalization layers after/before LSTM units to evaluate their impact on overall model performance. Specifically, we seek to identify whether batch normalization mitigates issues like vanishing or exploding gradients, ultimately leading to improved generalization of our models.
b. Layer Normalization: Proposed by Ba et al. (2016) [6], layer normalization offers an alternative normalization technique that normalizes activations across features, potentially improving model generalization and performance. Layer normalization operates across the feature dimension, which corresponds to the features dimension in the input data. In the context of time series classification, layer normalization normalizes activations independently for each time step across different features. Unlike batch normalization, layer normalization does not calculate normalization statistics across the mini-batch. Instead, it computes statistics independently for each example in the mini-batch, which may help preserve temporal dependencies and patterns. Furthermore, since layer normalization operates independently for each time step, it may be more suitable for preserving the temporal dynamics of the time series data and capturing feature-wise variations.
c. Masking: Masking layers are crucial for handling input sequences with padding, as they ignore padded values during training. By experimenting with and without masking layers, we aim to determine their effectiveness in improving model efficiency and accuracy by disregarding irrelevant padded values during computations. Furthermore, research suggests that masking can improve model efficiency and accuracy by focusing on relevant temporal patterns in the input data. This investigation will help us ascertain whether masking contributes to more robust time series classification models.
d. Dropout: Dropout involves randomly dropping units during training to prevent overfitting [7]. By adding dropout layers after LSTM and/or Dense units and varying the dropout rates, we intend to examine its impact on model generalization. Specifically, we aim to discern the optimal dropout rate and placement within our architecture to enhance the model's ability to generalize to unseen data while avoiding overfitting, especially in scenarios with limited training data or complex temporal dependencies.
e. L1/L2 Regularization: We experimented with different regularization strengths (lambda values) in the Dense layers. L2 regularization penalizes large weights in the network, promoting smoother weight distributions and reducing overfitting, while L1 regularization encourages sparsity in the weights, resulting in more interpretable models.
f. Single/Stacked LSTM Layers: In addition to experimenting with individual layers, we will investigate the influence of the configuration of LSTM layers on classification performance. We will explore both single-layer and multiple-layer LSTM architectures, evaluating their trade-offs in terms of model complexity and performance. By comparing the performance of models with different depths of LSTM layers, we seek to determine whether deeper architectures improve the model's capacity to capture long-term dependencies and extract relevant features from input time series data.
g. Bidirectional LSTM Layers: Bidirectional layershave the potential to capture temporal dependencies from both past and future time steps. Bidirectional LSTM layers offer the potential to improve model performance by incorporating information from both preceding and subsequent time steps. Through systematic experimentation with bidirectional layers, we aim to assess their efficacy in enhancing model accuracy and robustness in time series classification tasks.

Our final LSTM model is as seen on Fig. 2. Additionally,



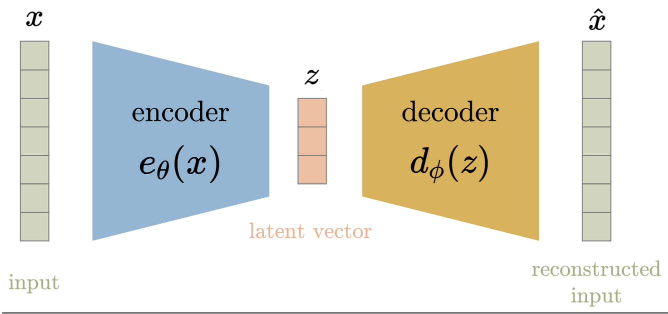
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 masking (Masking)           (None, 600, 8)            0

 lstm (LSTM)                 (None, 128)               70144

 dropout (Dropout)           (None, 128)               0

 dense_1 (Dense)             (None, 32)                4128

 dense_2 (Dense)             (None, 16)                528

 output (Dense)              (None, 1)                 17

=================================================================
Total params: 74817 (292.25 KB)
Trainable params: 74817 (292.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Fig. 2. LSTM model architecture

due to the class imbalance, we trained our final tuned model twice: once on the original imbalanced dataset, and once on an oversampled dataset where trips in the minority class are randomly duplicated to match the size of the majority class.

E. AutoEncoders

1) Typical AutoEncoder (AE): The typical AE is a neural network that learns to encode its' input data into a smaller dimensional latent space, and then reconstruct the output back to the original input. The architecture is comprised of 3 main components:

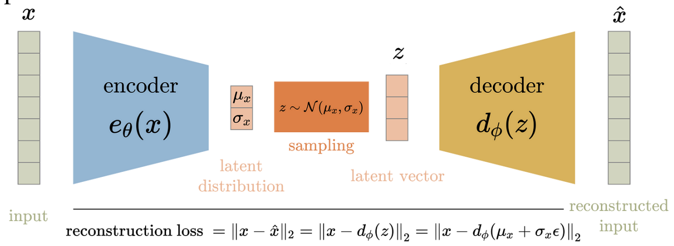


$$loss = \|x - \hat{x}\|_2 = \|x - d_\phi(z)\|_2 = \|x - d_\phi(e_\theta(x))\|_2$$

a. Encoder: Built using LSTM layers followed by Dense layers. The LSTM layers capture temporal dependencies essential to maintain the sequential integrity of the GPS data.
b. Latent Space: A high dimensional vector representing the encoded version of the input data. This compressed representation is then used by the decoder to reconstruct the input data, bringing out its most prominent features
c. Decoder: Reconstructs input data from the latent vector representation.

The reconstruction loss used is the mean squared error (MSE) between the encoder input and the decoder's output. A higher error suggests that the input pattern is unusual from the patterns learned during training.

2) Variation AutoEncoder (VAE): The VAE extends the capabilities of the standard encoder with a probabilistic approach. The architecture includes:



$$\text{reconstruction loss} = \|x - \hat{x}\|_2 = \|x - d_\phi(z)\|_2 = \|x - d_\phi(\mu_x + \sigma_x \epsilon)\|_2$$

$$\mu_x, \sigma_x = e_\theta(x), \quad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$$

$$similarity\ loss = KL\ Divergence = D_{KL}(\mathcal{N}(\mu_x, \sigma_x) \parallel \mathcal{N}(\mathbf{0}, \mathbf{I}))$$

$$loss = reconstruction\ loss + similarity\ loss$$

a. Encoder: Just like the AE, LSTM layers are also used. However, instead of a fixed latent space representation, it learns to output 2 parameters: mean (μ) and variance (σ^2). These parameters are used to describe the underlying probability distribution (usually Gaussian) where the latent variables are sampled from.
b. Latent Space: The sampling layer uses the distribution parameters to generate latent variables. Its stochastic nature introduces variability in the encoding, making the model more robust, enhancing its' ability to generalise to new unseen data.
c. Decoder: Reconstructs input data from the sampled latent variables. The reconstruction error is combined with the Kullback-Leibler divergence (KL-divergence) to form the loss function

3) Detection Strategy: Both models utilise the reconstruction error as the principal metric for identification. By training our AE on non-fraudulent GPS trips, the model learns the typical patterns of non-fraudulent trips. If our hypothesis that faked GPS signals have a higher inaccuracy is true, when the model is then presented with fraudulent trips, it should in turn have a generally higher reconstruction errors, indicating potential fraud.

The threshold for this error is set by us, and can be dependent on the data. Usual strategies include setting the 99th or 95th percentile of non-fraud reconstruction error, or taking the mean and adding 1 to 2 standard deviations to it. In our classification, we assume a normal distribution of reconstruction errors, and will set the threshold as follows:

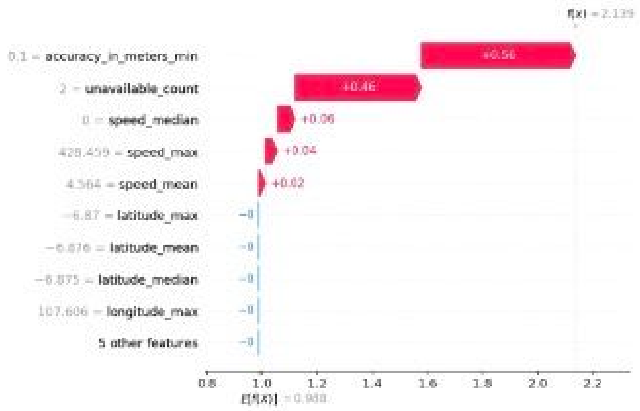
$$\begin{aligned}\text{threshold} = &\mu_{\text{Non-Fraud Reconstruction Error}} \\ &+ 2\sigma_{\text{Non-Fraud Reconstruction Error}}\end{aligned}$$

F. *Additional Notes on Model Evaluation*

Please refer to Table VI for more information on results based on evaluation metrics, rationale, and lessons learnt.

G. *Additional Notes on Error Analysis*

1) Logistic Regression: In our error analysis of the Logistic Regression model, we took a comprehensive approach to uncover discrepancies in predictions and identify opportunities for enhancing the model. We started by analyzing the feature importance plot to observe how each feature impacts the model's predictions. This examination helped us understand which features were crucial in the decision-making process of the model.



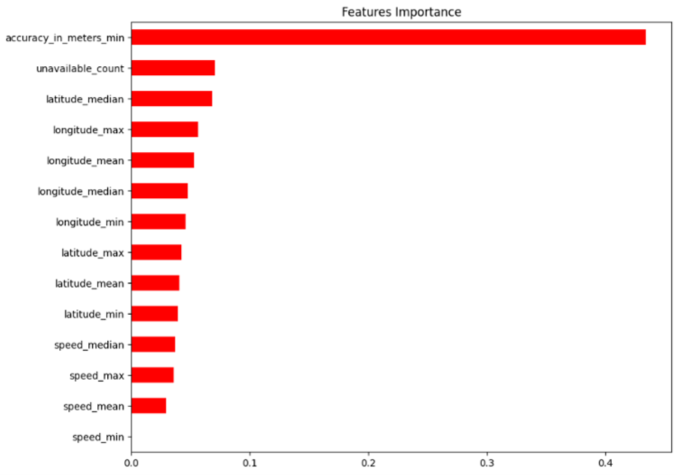
In our detailed analysis of the Logistic Regression model, we employed SHAP (SHapley Additive exPlanations) force plots as a sophisticated tool to pinpoint the features that play a crucial role in the model's prediction accuracy. This technique enabled us to visually decompose each prediction and illustrate the impact of each feature on the model's output. By utilizing SHAP force plots, we could clearly see which features contributed positively or negatively to the final decision, providing a granular view of the prediction dynamics. The SHAP force plots served not only to highlight the significant contributors but also to offer insights into the relative strength and direction of each feature's influence. This approach was instrumental in

TABLE VI
MODEL EVALUATION SUMMARY

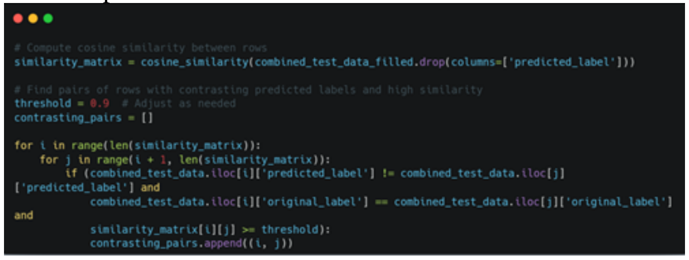
Model	Accuracy	Recall	BCE	AUC	Rationale	Lesson Learnt
Logistic Regression (Baseline)	71.4%	88.3%	0.56	0.79	Simplicity & interpretability, performance benchmark, fast and efficient, regularization and overfitting.	Sensitive to imbalanced data, importance of regularization, baseline for more complex models.
XGBoost Classifier (With SMOTE)	84.1%	84.5%	5.73	0.84	Ensemble learning technique, strong classifier, gradient based optimization, built-in regularization techniques.	Increased model complexity leading to overfitting resulted in higher BCE. Capacity to encapsulate the semantic intricacies inherent in sequential coordinates and geospatial data remains limited.
Temporal CNN	77.5%	83.4%	0.46	0.82	TCNNs can compute in parallel, hence being faster computationally.	Simpler model with parameters while having regularization tends to perform best.
LSTM (Single Layer)	81.8%	84.1%	0.40	0.87	LSTM is suitable for the time-series nature of dataset	Simpler model with fewer parameters and regularization tend to perform best as there is less overfitting.
AutoEncoder	70.0%	0.0048%	2.04	0.60	Autoencoders are great in detecting anomalies from input data.	Real and Faked GPS trips have similar GPS Signals, and hence may not be useful to detect anomalies

identifying key features that consistently impact the model's predictions across various instances, thereby offering a direct pathway to refine feature selection and potentially enhance model performance. These insights are critical for fine-tuning our model to reduce prediction errors and improve overall accuracy.

2) XGBoost Classifier: In our error analysis on the classical models, more specifically, the XGBoost model, we adopted a multi-faceted approach to gain insights into prediction inconsistencies and potential areas for model improvement. Initially, we examined the feature importance plot to visualize the relative significance of each feature in influencing the model's predictions. This analysis provided valuable insights into the pivotal features driving the model's decision-making process.

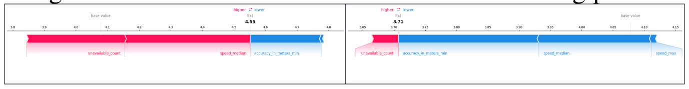


We then utilized cosine similarity to identify similar rows in our dataset, focusing on instances where predicted labels differed despite identical ground truth labels. By comparing feature vectors of all pairs of data points and setting a threshold for cosine similarity, we pinpointed potential inconsistencies in the model's decision-making process. This approach facilitated the identification of specific instances where the model may have struggled to generalize effectively or where certain features led to prediction errors, informing strategies for improvement and enhancing our understanding of the model's performance.



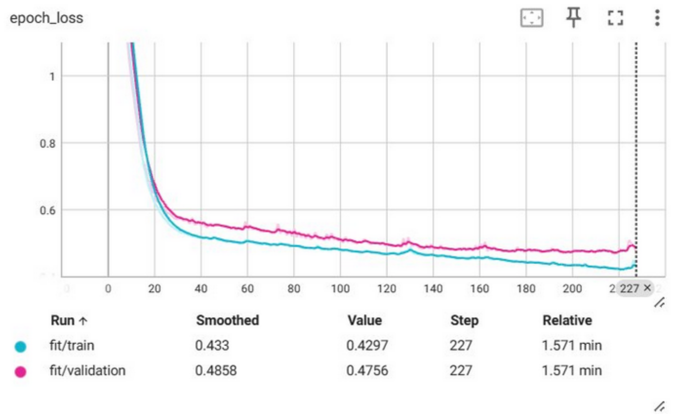
```
# Compute cosine similarity between rows
similarity_matrix = cosine_similarity(combined_test_data_filled.drop(columns=['predicted_label']))

# Find pairs of rows with contrasting predicted labels and high similarity
threshold = 0.9  # Adjust as needed
contrasting_pairs = []

for i in range(len(similarity_matrix)):
    for j in range(i + 1, len(similarity_matrix)):
        if (combined_test_data.iloc[i]['predicted_label'] != combined_test_data.iloc[j]
['predicted_label'] and
            combined_test_data.iloc[i]['original_label'] == combined_test_data.iloc[j]['original_label']
and
            similarity_matrix[i][j] >= threshold):
            contrasting_pairs.append((i, j))
```

Following the identification of similar rows, we leveraged SHAP (SHapley Additive exPlanations) values to further dissect the contribution of each feature to individual predictions. Through SHAP analysis, we visualized the impact of features on prediction probabilities, discerning which features exerted the greatest influence on the model's decision-making process.



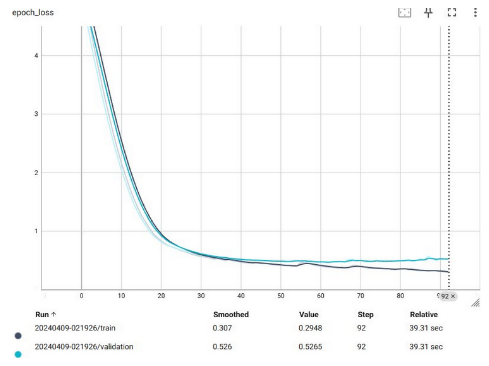
The SHAP analysis provides crucial insights into the XGBoost model's prediction behavior by indicating the features that predominantly influence its predictions. Higher SHAP values suggest features that push towards predicting label 1, while lower values indicate features leaning towards label 0. Notably, in our analysis, a higher unavailable_count had a pronounced impact on predicting label 1, suggesting an association of increased unavailable pings with this label, while a lower accuracy_in_meters_min was more influential in predicting label 0, indicating lower GPS accuracy correlating with this label. By integrating SHAP analysis, we gained actionable insights into feature importance, aiding in model interpretation and identifying potential areas for refinement.

3) Neural Network Models: First, we plotted our BCE losses over time using TensorBoard to ensure that the validation set loss values have converged to their optimal values. Referring to Figures 3 and 4, we can see that there is a slight increase in loss after a certain time, indicating that the model has achieved convergence.



	Run ↑	Smoothed	Value	Step	Relative
●	fit/train	0.433	0.4297	227	1.571 min
●	fit/validation	0.4858	0.4756	227	1.571 min

Fig. 3. TCNN BCE loss over epoch



	Run ↑	Smoothed	Value	Step	Relative
●	20240409-021926/train	0.307	0.2948	92	39.31 sec
●	20240409-021926/validation	0.526	0.5265	92	39.31 sec

Fig. 4. LSTM BCE loss over epoch

Furthermore, to calculate average absolute weight of each feature for feature importance, let $\mathbf{W}$ be the matrix of weight into the first layer of our neural network. For example, in our final LSTM model of 128 units, $\mathbf{W} \in \mathbb{R}^{C \times 512}$ as we have C features (V-D2) and $128 \times 4 = 512$ separate units due to the presence of input, forget, output, and cell state gates. Afterwards, we take the modulus of each weight in the matrix, scale, and average them. Then, the feature importance imp_c for feature c is

$$\text{imp}_c = \frac{\sum_{i=1}^{512} |\mathbf{W}_{c,i}|}{512}.$$

The feature importance plot is as seen in Fig. 5. Next, we plotted confusion matrices for each model to understand the proportion of misclassification in each class. Using our LSTM model as an example in Fig. 6, 33.3% of predicted fake trips are in fact genuine, which might lead to unfairness and reduced trust as honest drivers are flagged with fake trips. Furthermore, the model has a high false positive rate (FPR) of 23.5% which means that almost one-quarter of overall fake trips are not detected as fake. This prompted us to visualize misclassified trips to investigate the model's shortcomings.

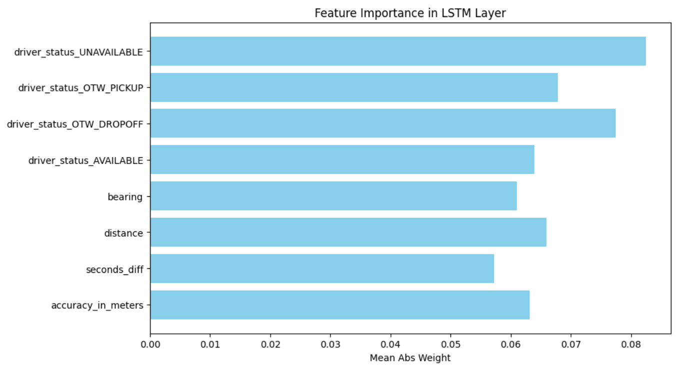


Fig. 5. Feature importance plot for neural networks

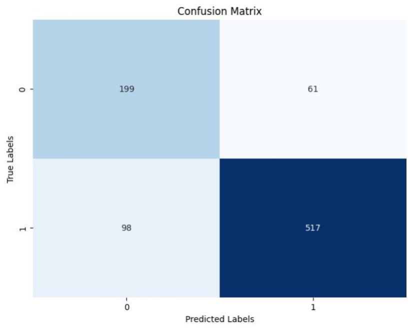


Fig. 6. LSTM model confusion matrix

We observed that there is a lot of regularity in fake trips, which might lead the model to misclassify them as genuine (Figs.7 8 and 9).

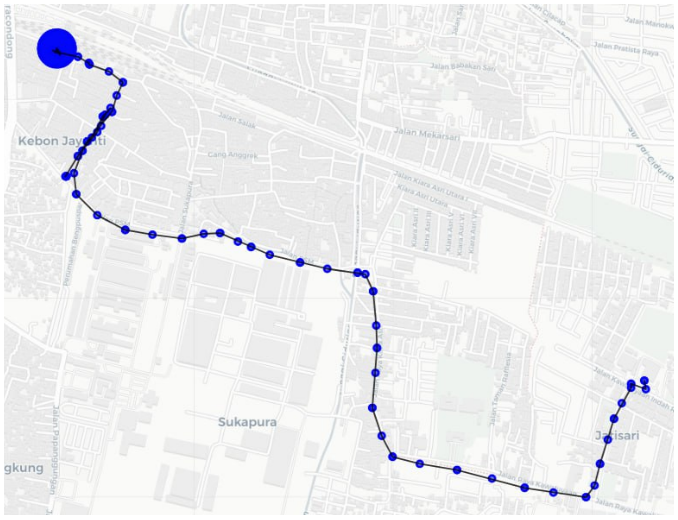


Fig. 7. Plot of trip F1000, misclassified as genuine

Furthermore, there were also instances where the model did not recognise large coordinate jumps, as seen in Fig. 10. The model did not recognise that the driver jumped between two different roads, which a human would observe as a spoofing behaviour.

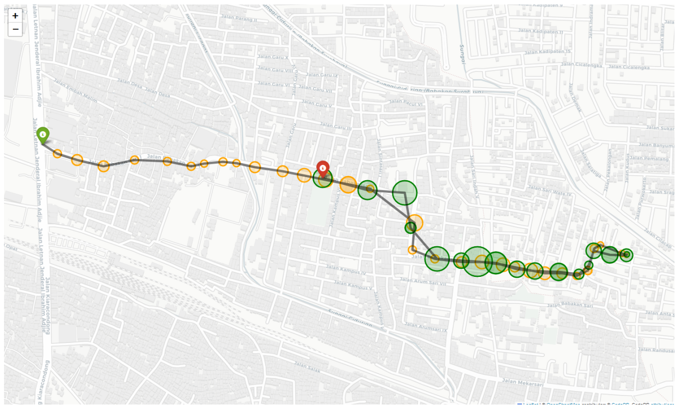


Fig. 8. Plot of trip RB284, misclassified as genuine

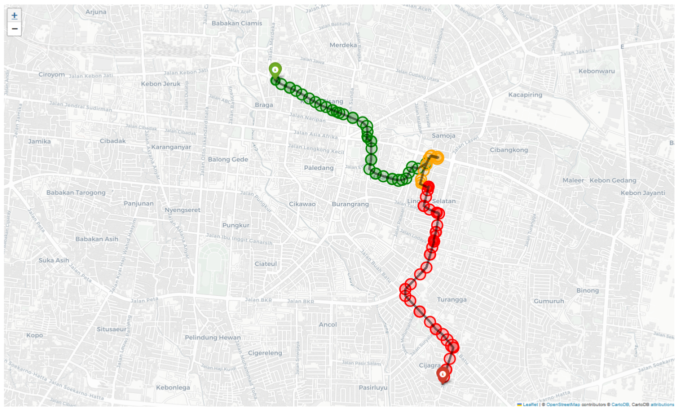


Fig. 9. Plot of trip RB661, misclassified as genuine

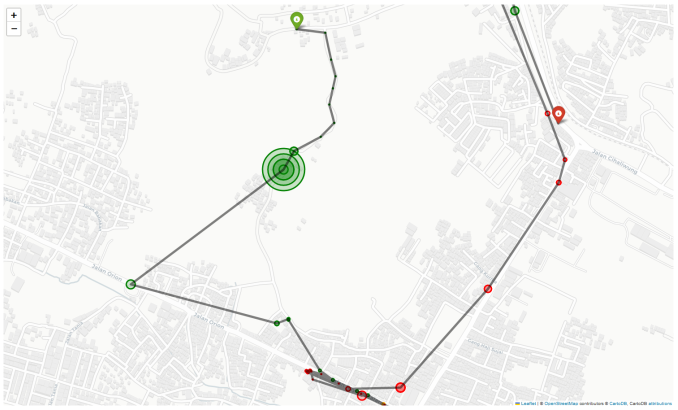


Fig. 10. Plot of trip RB994, misclassified as genuine

The AE on the other hand, can identify large coordinate jumps, as seen in 11 where the driver briefly flew to Antarctica. AE is also able to correctly detect trips with **extremely** large inaccuracies such as in 12, which has an inaccuracy of over 2km.

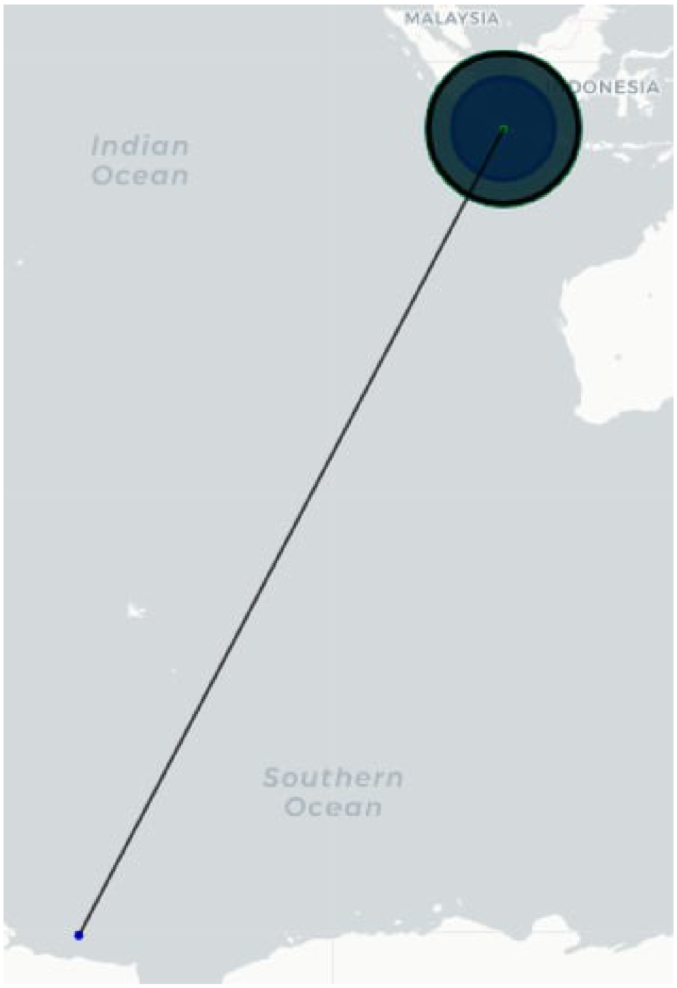


Fig. 11. Plot of trip F1521, correctly classified as fake

However, this doesn't explain the low recall values in AEs. Upon visualising the misclassified trips, a better understanding is reached. We observe that not only does genuine trips also contain unrealistic coordinate jumps (13, they also consists of trips with multiple large (albeit not extreme) inaccuracies (14 with inaccuracy of ¿1km).

To add on to the difficulty, we found that quite a number of fraudulent trips have generally clean and regular GPS Signals, which even to humans, looks genuine and not faked. For example, in 15, the trip looks very regular and similar to a genuine trip, but yet, this is a fraudulent trip. This means that our hypothesis that faked GPS trips have a higher reconstruction loss to be false.

Further exploring on this, we overlaid the reconstruction losses between fraudulent and non-fraudulent trips to confirm

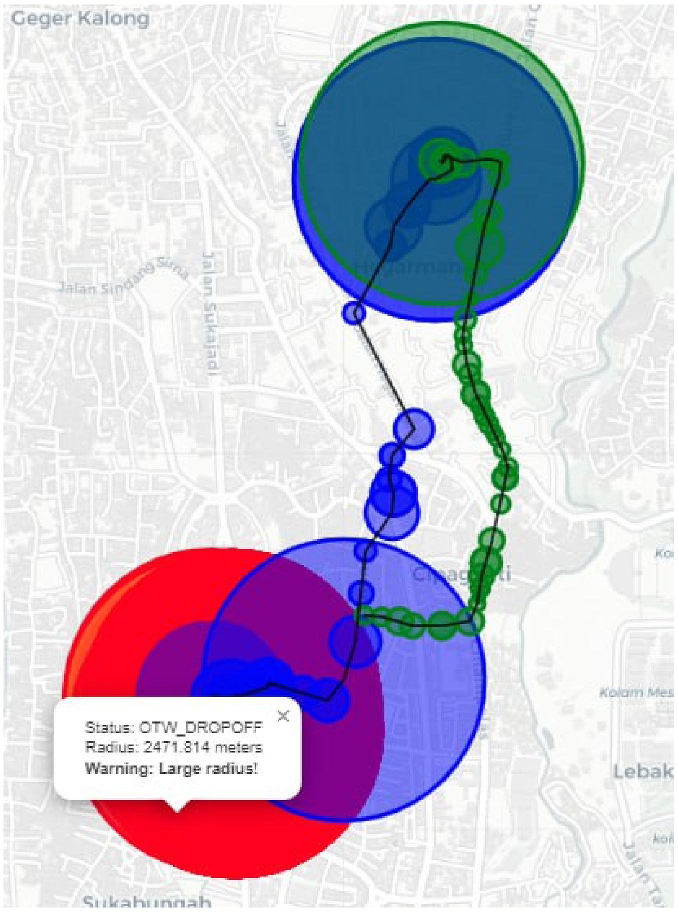


Fig. 12. Plot of trip F206, correctly classified as fake

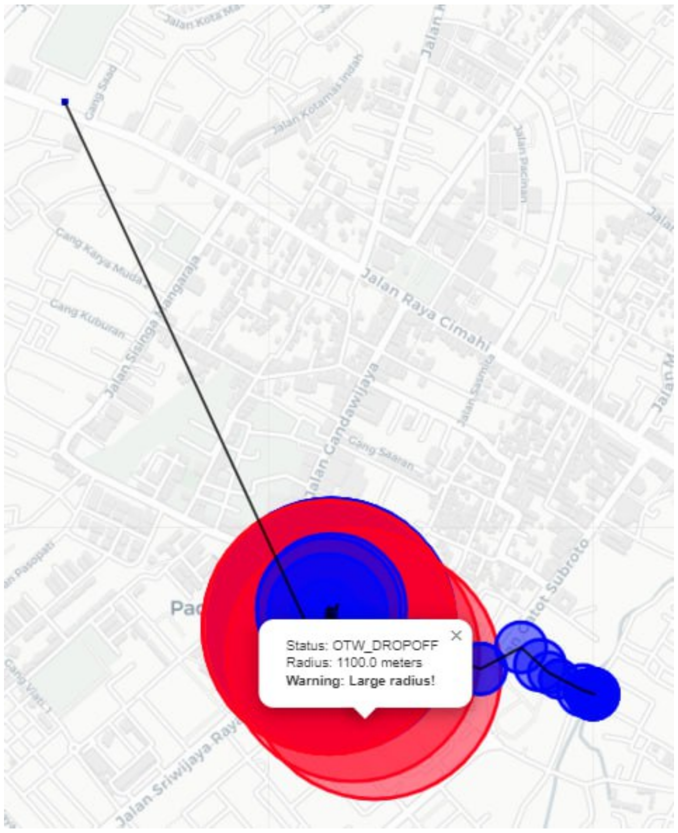


Fig. 13. Plot of genuine trip with an unrealistic coordinate jump

that hypothesis to be false. As shown in 16, we observe that the distribution in reconstruction loss between both fake and genuine trips to be very similar, showing that both fraudulent and genuine trips to have similar GPS signal patterns. Although, we can observe that all trips with unusually large reconstruction losses are fraudulent, and the AE is able to correctly identify them.

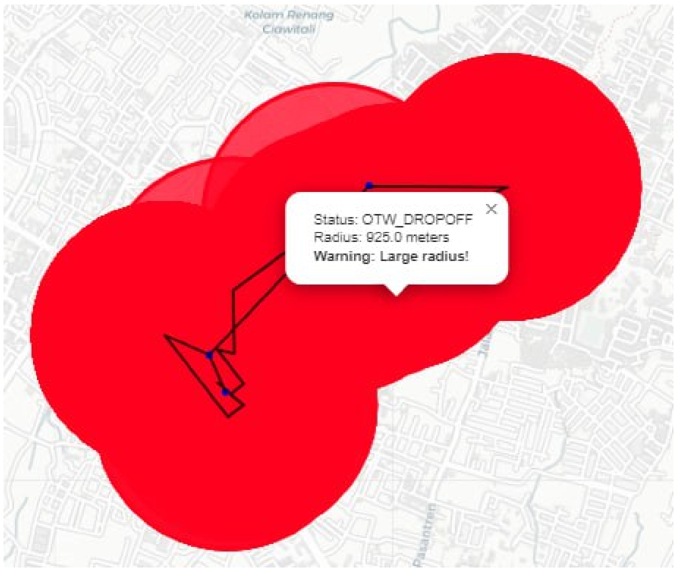


Fig. 14. Plot of genuine trip multiple large inaccuracies

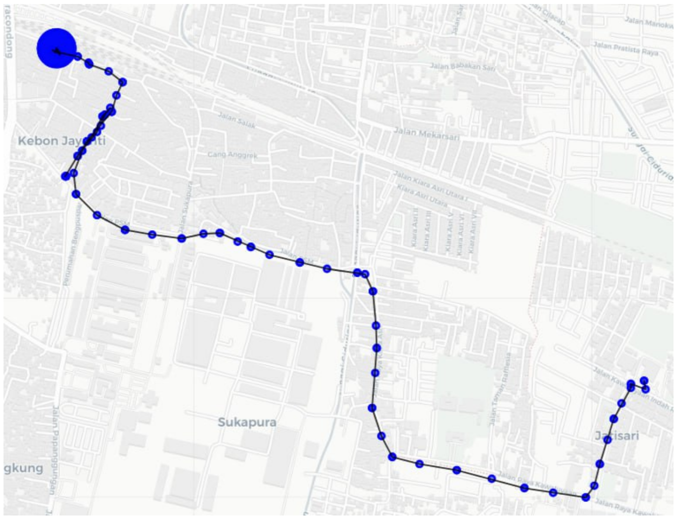


Fig. 15. Plot of genuine trip multiple large inaccuracies

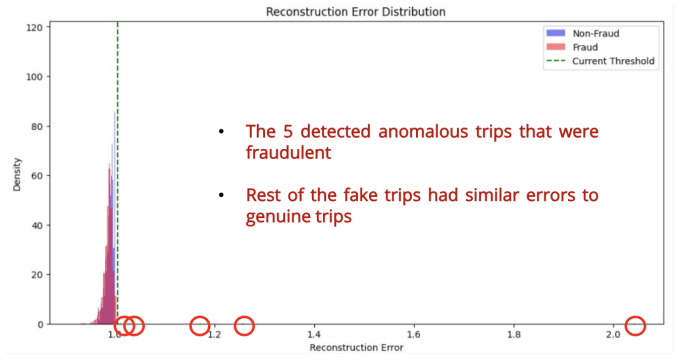


Fig. 16. Plot of the distribution of reconstruction losses between fraudulent and non fraudulent trips