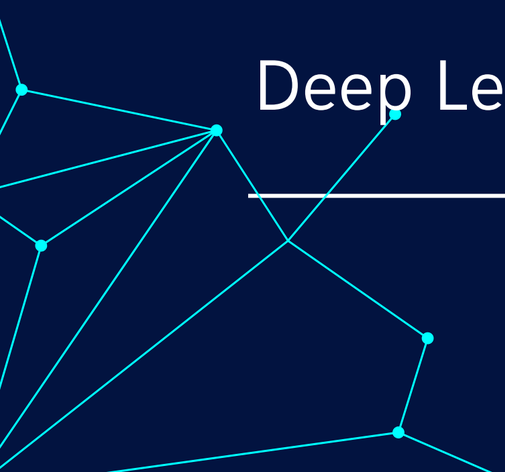
DETECTING MALICIOUS URLS

SMART URL FILTERING WITH ML &
DL MODELS

- BENGHENIMA Hafsa
- GHANDOUZ Amina

Presentation Plan



Problem Statement

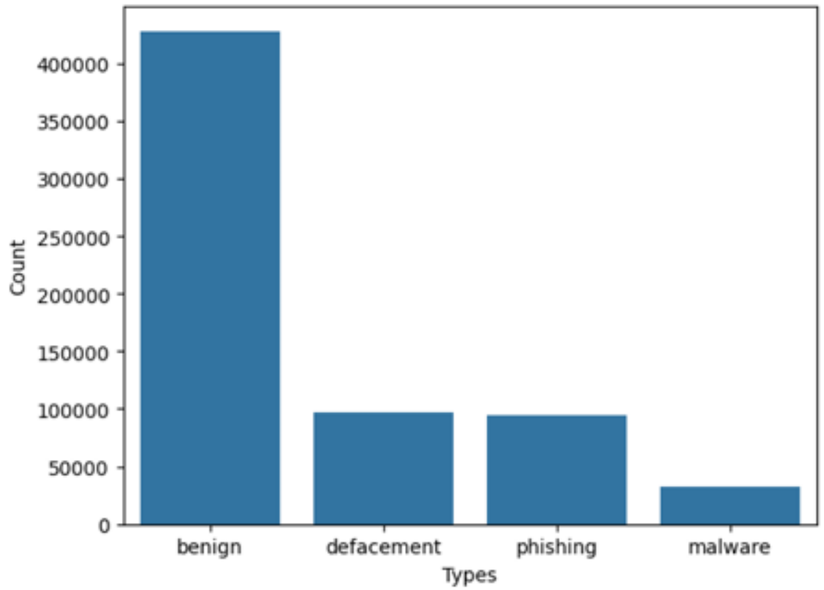
- The main objective of this project is to build a system that can automatically classify URLs as benign or malicious.
- We compare several machine learning models and a deep learning model (LSTM) to identify the most effective approach.

Dataset Description

- **Source:** Kaggle
- **Number of rows:** 651191

	url	type
0	br-icloud.com.br	phishing
1	mp3raid.com/music/krizz_kaliko.html	benign
2	bopsecrets.org/rexroth/cr/1.htm	benign
3	http://www.garage-pirenne.be/index.php?option=...	defacement
4	http://adventure-nicaragua.net/index.php?optio...	defacement

- **Distribution of classes:**



Traditional Machine Learning Approach

1. Preprocessing

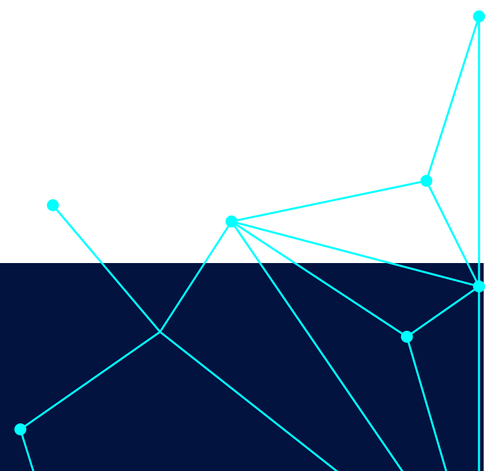
1. Remove Duplicate URLs
2. Clean URLs
3. Encode URL Types

3. Data Splitting

70% Training / 30% Testing

2. Feature extraction

1. URL Length
2. Character Counts:
3. Use of URL Shorteners
4. Abnormal Structure
5. HTTPS Check
6. IP Address in URL
7. Suspicious Characters



4. ML Models Used

1. Decision Tree Classier
2. Random Forest Classier
3. KNN
4. SGD Classifier
5. Gaussian Naive Bayes

Model Performance Summary :

Model	Accuracy	Precision	Recall	F1-Score
Decision Tree	90.6%	0.90	0.91	0.90
Random Forest	91.52%	0.91	0.92	0.91
KNN	89.42%	0.89	0.89	0.89
SGD	82.07%	0.81	0.82	0.76
Gaussian NB	80.18%	0.78	0.80	0.75

Identifying and Addressing Class Imbalance

- After evaluating the baseline models, we noticed that the dataset was imbalanced.This imbalance negatively affected model performance, especially for the minority classes.
- To address this, we applied two balancing techniques:

1. SMOTE

Creates synthetic examples for the minority class

Model	Accuracy
Decision Tree	85.51%
Random Forest	86.10%
KNN	83.73%
SGD Classifier	60.87%
Gaussian Naive Bayes	80.02%

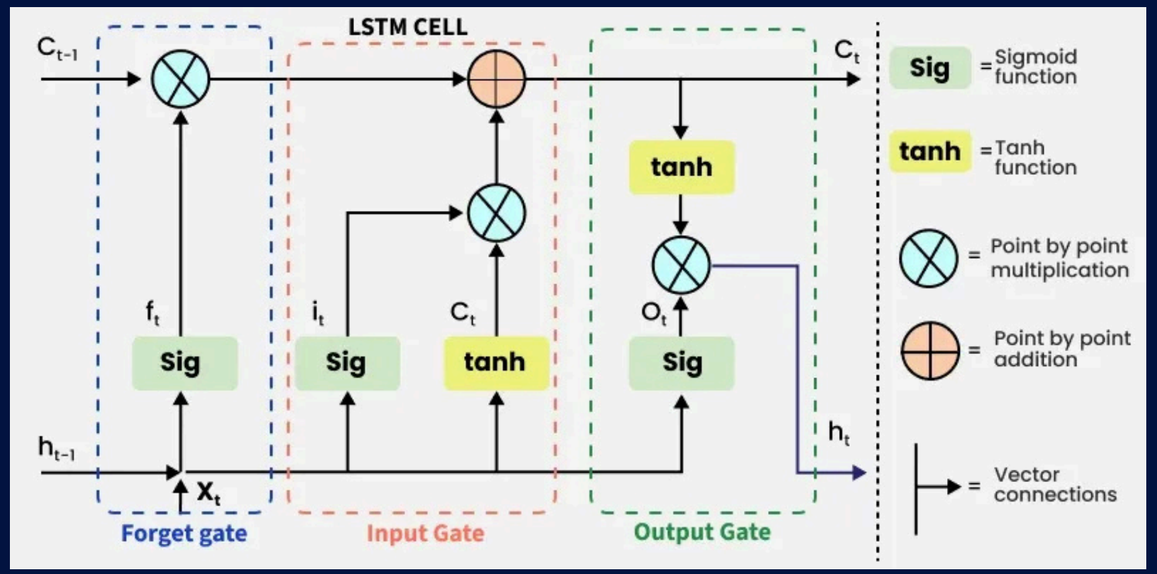
2. Undersampling

Decrease the number of samples to be exactly like the minority class.

Model	Accuracy
Decision Tree	81.89%
Random Forest	84.12%
KNN	79.23%
SGD Classifier	76.69%
Gaussian Naive Bayes	80.23%

Deep Learning Approach

- Why LSTM?
  - LSTMs (Long Short-Term Memory networks) are effective for sequential data like text and URLs.
  - They can capture character-level patterns and contextual sequences in URLs



- **Model Architecture**

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	?	0 (unbuilt)
lstm_5 (LSTM)	?	0 (unbuilt)
dropout_7 (Dropout)	?	0 (unbuilt)
dense_6 (Dense)	?	0 (unbuilt)
dense_7 (Dense)	?	0 (unbuilt)

Total params: 0 (0.00 B)

Trainable params: 0 (0.00 B)

What Was Improved?

1. Bidirectional LSTM Layers
   Captures both forward and backward URL patterns
2. Dropout layers added and applied after each LSTM and Dense layer
3. Increased output_dim to 64, that provides richer character-level representation of URLs
4. Increase number of epoches to 20

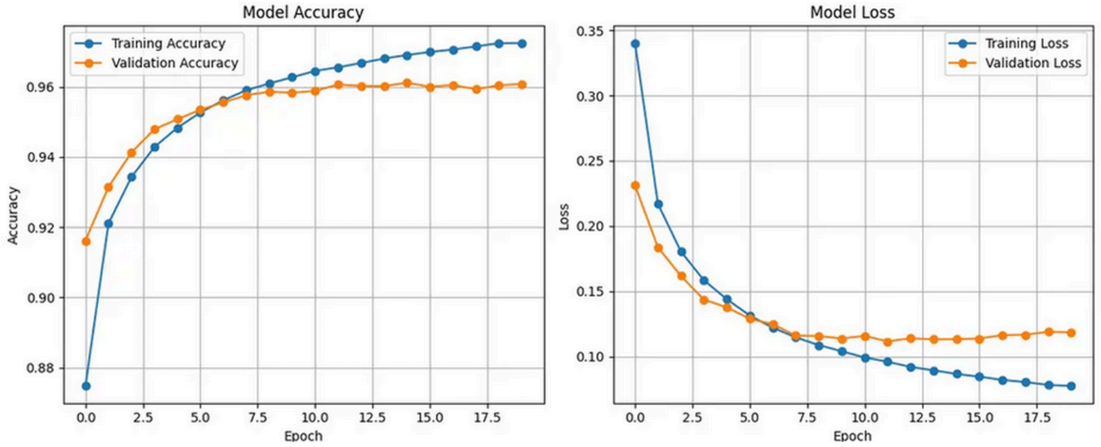
- Compilation :
  Loss : categorical_crossentropy
  Optimizer : adam.
  Metrics : Accuracy.

Class Imbalance and LSTM

- Unlike traditional ML models, LSTM was trained without applying SMOTE or oversampling.
- Reason: These techniques in sequences can introduce noise or artificial patterns, which may harm generalization.

LSTM Performance Results

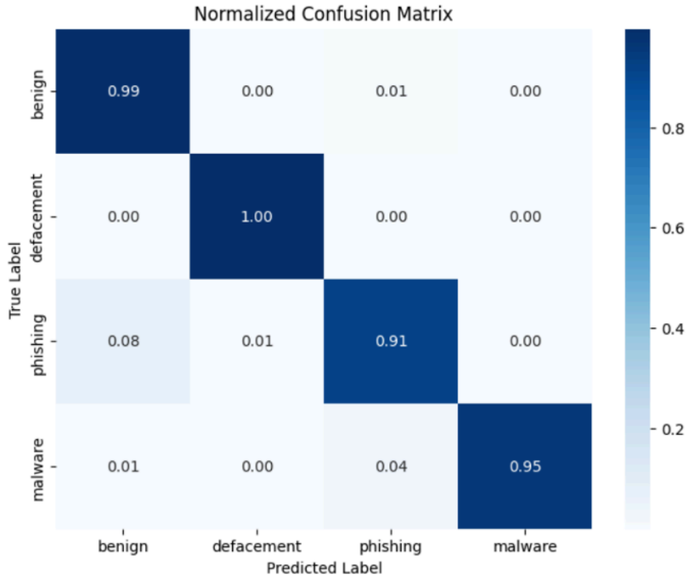
- **Plots:**



- **Classifcation Report:**

	precision	recall	f1-score	support
benign	0.98	0.99	0.99	85778
defacement	0.99	1.00	0.99	19104
phishing	0.95	0.91	0.93	18836
malware	0.99	0.95	0.97	6521
accuracy			0.98	130239
macro avg	0.98	0.96	0.97	130239
weighted avg	0.98	0.98	0.98	130239

- **Confusion Matrix:**



Testing the best model

After evaluating the model on the test set, we tested it on a new, unseen URL to observe how it performs in a real-world scenario.

🧪 **Input URL:**
http://192.168.4.1/files/install.exe
🧠 **Steps:**

1. Tokenization using the original tokenizer
2. Padding to max length of 100
3. Prediction using the trained LSTM model

📢 **Result:**
✅ The URL is classified as: ⚠️ Malware

🧪 **Input URL:**
www.google.com
🧠 **Steps:**

1. Tokenization using the original tokenizer
2. Padding to a max length of 100
3. Prediction using the trained LSTM model

📢 **Result**:
✅ The URL is classified as: Benign

- **In jupyter:**

```
# Example URL
new_url = ["http://192.168.4.1/files/install.exe"]

# Tokenize with the ORIGINAL tokenizer (do not re-fit!)
X_seq = tokenizer.texts_to_sequences(new_url)  # Use the same tokenizer from training

# Pad sequences (same maxlen=100 and padding='post' as during training)
X_pad = pad_sequences(X_seq, padding='post', maxlen=100)

# Predict
y_pred_probs = model.predict(X_pad)
y_pred_class = np.argmax(y_pred_probs, axis=1)
predicted_label = target_names[y_pred_class[0]]
print(f"The URL is classified as: {predicted_label}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
The URL is classified as: malware
```

```
# Example URL
new_url = ["www.google.com"]

# Tokenize with the ORIGINAL tokenizer (do not re-fit!)
X_seq = tokenizer.texts_to_sequences(new_url)  # Use the same tokenizer from training

# Pad sequences (same maxlen=100 and padding='post' as during training)
X_pad = pad_sequences(X_seq, padding='post', maxlen=100)

# Predict
y_pred_probs = model.predict(X_pad)
y_pred_class = np.argmax(y_pred_probs, axis=1)
predicted_label = target_names[y_pred_class[0]]
print(f"The URL is classified as: {predicted_label}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
The URL is classified as: benign
```

THANK YOU