Project Plan: Multi-Class Malicious URL Classifier

Part 1: Project Proposal

What are some medium level NLP projects I could do for an intro to deep learning course?

(give the response you gave before but now for a multi-class classification project for malicious url detection)

The Concept

Most basic security filters just tell you if a website is "Safe" or "Unsafe" (Binary Classification). In this project, you will build a Deep Learning model that acts as an intelligent security analyst. It won't just block a URL; it will read the URL string as text and categorize exactly *what kind* of threat it poses.

The Goal

Build a model that takes a raw URL string as input and classifies it into one of four categories:

1. **Benign:** Safe, legitimate traffic.
2. **Phishing:** Social engineering sites trying to steal credentials (e.g., paypal-login-secure.com).
3. **Malware:** Links hosting viruses or ransomware payloads (e.g., .../download/trojan.exe).
4. **Defacement:** Legitimate sites that have been hacked and altered by attackers.

Why this is "Medium" Level

- **It's pure NLP:** You aren't calculating features like "URL length" manually. You are feeding raw text to the model and forcing it to learn the grammar of the internet.
- **Data Complexity:** Real-world security data is messy and highly imbalanced (there are way more safe sites than malware). You will have to learn techniques like **undersampling** or **class weighting**.
- **Architecture:** You will implement standard Sequence Models (RNNs/LSTMs) but applied to a non-standard language (URLs instead of English sentences).

The Architecture (How you apply Deep Learning)

You will treat the URL as a sequence of characters or words.

1. Input Processing (NLP Pipeline):

- **Tokenization:** You will experiment with breaking the URL into **Characters** (good for catching obfuscation like paypa1) or **Words** (good for catching semantic context like login + secure).
- **Padding:** You will standardize inputs so the model can process batches (e.g., padding all URLs to length 50 or 138).

2. The Model Layers:

- **Embedding Layer:** A learned lookup table that converts your integer tokens into dense vectors. This allows the model to learn that paypal and chase are related concepts (banks), while http and https are related concepts (protocols).
- **Bidirectional LSTM (Long Short-Term Memory):** The core "brain." It reads the URL forwards and backwards. It learns to connect a suspicious domain at the *start* of the string with a suspicious file extension at the *end* of the string.
- **Dense Classification Head:** A final fully connected layer with **4 neurons** and a **Softmax** activation function to output the probability for each of the 4 classes.

Key Learning Objectives

By the end of this project, you will understand:

1. **Sequence Modeling:** How to use LSTMs/GRUs for text classification.
2. **Multi-Class Classification:** How to move from Binary Cross-Entropy (0/1) to Categorical Cross-Entropy (One-hot encoding).
3. **Handling Imbalanced Data:** How to ensure your model detects rare threats (like Malware) and doesn't just guess "Benign" every time.
4. **Evaluation Metrics:** Why "Accuracy" is a bad metric for security, and why you must use **Precision, Recall, and F1-Score**.

Part 2: Project Roadmap

Can you breaks project into phases (i.e. data acquisition, tokenization, etc.)?

💻 Phase 1: Data Acquisition and Preprocessing (The NLP Foundation)

Goal: Convert millions of raw URL strings into clean, balanced, numerical matrices that a deep learning model can read.

- **1.1 Data Acquisition:**
  - **Action:** Write a script (data_acquisition.py) to programmatically fetch data from four sources: **Kaggle** (Base), **URLhaus** (Malware), **OpenPhish** (Phishing), and **Tranco** (Benign).
  - **Key Logic:** Deduplicate all lists and implement **undersampling** to ensure the "Benign" class doesn't drown out the malicious ones.
  - **Output:** A balanced final_dataset.csv with ~220k URLs.
- **1.2 Vocabulary Creation:**

    - **Action:** Scan the entire dataset to build a "dictionary."
    - **Word-Level:** Identify the top 50,000 most common URL "words" (tokens).
    - **Char-Level:** Identify every unique character (e.g., a, b, /, .).
    - **Output:** word_vocab.json and char_vocab.json.
- **1.3 Tokenization & Encoding:**
    - **Action:** Translate the text URLs into numbers using your vocabularies.
    - **Example:** http://google.com $\rightarrow$ [12, 45, 88, 101...].
    - **Output:** Intermediate encoded lists.
- **1.4 Sequence Padding:**
    - **Action:** Standardize input lengths so the model can process them in batches.
    - **Word-Level:** Pad/truncate all sequences to **50** tokens.
    - **Char-Level:** Pad/truncate all sequences to **138** characters.
    - **Output:** X_data.npy (Features) and y_data.npy (Labels).
- **1.5 Data Split:**
    - **Action:** Divide the data into three sets using a **stratified split** to maintain class balance.
    - **Output: Training Set** (70%), **Validation Set** (15%), and **Test Set** (15%).

🧠 Phase 2: Deep Learning Model Construction (The Architecture)

Goal: Define the "brain" of the project—the neural network architecture that will learn to recognize threats.

- **2.1 Embedding Layer (The Translator):**
    - **Action:** Define the first layer to convert integer IDs into dense, meaningful vectors (e.g., size 32 or 64). This allows the model to learn relationships between tokens.
- **2.2 Sequential Core (The Brain):**
    - **Action:** Implement a **Bidirectional LSTM** (Long Short-Term Memory) layer. This processes the sequence forwards and backwards to capture context.
- **2.3 Classification Head (The Judge):**
    - **Action:** Add fully connected (**Dense**) layers with **Dropout** (to prevent overfitting).
    - **Output:** A final layer with **4 neurons** (one for each class: Benign, Defacement, Phishing, Malware) using **Softmax** activation.
- **2.4 Model Compilation:**
    - **Action:** Configure the model with the **Adam** optimizer and **Sparse Categorical Cross-Entropy** loss function.

🚀 Phase 3: Training (The Learning Process)

Goal: Feed the data into the model so it can learn the patterns of malicious URLs.

- **3.1 Training Loop:**
    - **Action:** Feed the X_train data into the model in batches (e.g., 64 URLs at a time) for multiple "epochs" (passes through the dataset).
- **3.2 Validation Monitoring:**

    - **Action:** After every epoch, test the model on the X_val set. If accuracy stops improving, stop training early (**EarlyStopping**) to save time and prevent overfitting.
- **3.3 Checkpointing:**
    - **Action:** Automatically save the "best" version of the model (the one with the lowest validation loss) to a file.

📊 Phase 4: Evaluation and Analysis (The Verdict)

Goal: Test the models on unseen data and determine which approach (Word-Level vs. Character-Level) is superior.

- **4.1 Final Inference:**
    - **Action:** Load the saved models and run predictions on the **Test Set** (X_test), which the model has never seen before.
- **4.2 Metric Calculation:**
    - **Action:** Generate a **Classification Report**. Look beyond simple accuracy; focus on **Precision** (did we flag safe sites?) and **Recall** (did we miss malware?) for each class.
- **4.3 Confusion Matrix:**
    - **Action:** Create a visual grid showing exactly where the model made mistakes (e.g., "It confused 5% of Phishing URLs for Benign").
- **4.4 Comparative Analysis:**
    - **Action:** Compare the Word-Level Model vs. the Character-Level Model. Determine which one was better at spotting specific threats like Phishing vs. Malware.