Homework 1

Domain Generation Algorithms (DGA) Detection using NLP Techniques, LSTM, and CNN

Objective

This homework aims to:

- Apply **NLP preprocessing** techniques on DGA domain datasets.
- Implement **LSTM** and **CNN** models to classify domains as **benign or malicious**.
- Analyze, visualize, and document the results in a **technical report**.

Submission Requirements

You must submit the following:

1. **Python Code** including:
   - Data preprocessing
   - LSTM and CNN models
   - Evaluation and visualizations
2. **Technical Report** (Latex-based PDF) including:
   - Answers to open questions
   - Detailed description of your approach and techniques
   - Visualization diagrams with explanations
   - Comparative analysis between LSTM and CNN

Student Name:		**ID:**
Section	**Earned Points**	**Max Points**
Preprocessing and Data Analysis		20
LSTM Model Implementation and Evaluation		20
CNN Model Implementation and Evaluation		20
Visualizations and Technical Report Quality		20
Open Questions Answered Clearly		20
Total		**100**

Q1) After applying NLP techniques (such as tokenization, padding, and character embeddings) to preprocess the DGA dataset, explain:

• Why are these preprocessing steps necessary when using LSTM and CNN models?

• How do these steps help improve the model's ability to detect DGA domains?

(Answer in 250–400 words, using examples from your preprocessing.)

Text preprocessing is a necessary step when performing natural language processing tasks such as DGA detection. It helps us standardize the data and prepare it for processing by the model to provide useful results.

In our text preprocessing phase, we started with importing the dataset, then we encoded our label column 'isDGA' to either 0 (legitimate) or 1(DGA). This step is important because machine learning models, including CNN and LSTM, require that the input data is in a numerical format to enable the model to train on this data. Next, we ensured that our dataset is not imbalanced, and it turned out to be perfectly split between DGA and legitimate inputs (80,000 each). We had very few nulls and duplicates that we eliminated and because they are few, they have minimal effect on our overall data size.

Our next step was to lowercase the domain and host strings. This is a very important step in NLP tasks, because it reduces the noise in our data. It ensures that all our data is consistent and no uppercase letters confuse the model and interfere with its learning process.

Then, we performed character-based tokenization on our inputs. This enables the model to recognize sub words or creative spellings of words. In other terms, it avoids the OOV problem and ensures the model can effectively train without memorizing a huge vocabulary set. Also, we embedded the characters such as p being mapped to 15. The embedding gives closer characters such as 'a' and 'e' higher correlation than different characters such as 'a' and '9'.

Moreover, we used padding to standardize the host name length. After testing with the padding length, we decided to use 30 characters padded with zeros as it showed the best performance along with avoiding complexity.

By following all those steps, we ensure that data is consistent and standardized, which enables the model to focus on the more important data and avoid any additional noise which may affect our model's accuracy. After performing the preprocessing, our data is now prepared for training our models.

Q2) Suppose your trained LSTM model achieves 90% accuracy but struggles to detect some DGA families with high randomness in domain names (e.g., using many unusual characters).

• What are the possible reasons for this limitation?

• Suggest two methods (NLP-based or model-based) to improve the detection performance.

(Answer in 250–400 words, and justify your suggestions.)

LSTM relies on recurring data patterns to form memories, but the accuracy of our model may be hindered by a number of factors. One potential problem that may have hindered our LSTM model is having some special characters. As we perform character based tokenization, having special unusual characters may hinder the performance of the model. LSTM would not be able to recognize and effectively embed those rare occurrence characters, therefore demonstrating a potential reason for why our LSTM may have struggled with DGA families that have high randomness in their domain name. Another potential reason for poor performance of LSTM could be due to padding. Padding affects the length of the domain name which in some cases might eliminate some information that is important for our model. In our case, this is less likely to be a reason for the poor model performance as we tested with different padding techniques such as adding zeros at the end or using imported specialized padding libraries. In addition, after trying multiple padding lengths, we found that 30 provided better performance than other options. By optimizing this hyperparameter, we narrow down the reasoning for poor LSTM performance.

There are many approaches that could help us improve our model performance. One approach is using BPE encoding instead of character based encoding. BPE is a strong tokenization method that takes into account frequent sequences of characters when tokenizing the domain name. It helps us detect sub-words and effectively encode them as needed. BPE is useful in this case because some domain names may have variations of the same word which could help the model improve its performance. Character based tokenization is a standard approach, but it is more simplistic relative to BPE, which could be a problem in more complex models.

Data augmentation is another potential approach that could help us improve our model performance. Data augmentation means adding new data points by adding distortions or noise to existing data points. It allows us to synthetically generate domain examples from underrepresented DGA families. As we mentioned before, special characters may be a problem in domain names, so to improve the model performance, we could use data augmentation on inputs that have special characters or inputs that the model failed to classify effectively. This way, the model would get used to the special characters in domain names, and consequently would perform better.

1 Introduction

A **Domain Generation Algorithm (DGA)** is a programmatic routine used by malware to create hundreds—or even thousands—of pseudo-random domain names every day. By continuously rotating command-and-control (C2) endpoints, DGAs invalidate static blacklists and make coordinated takedowns extremely difficult.

We tackle the problem with a **two-stage** deep-learning pipeline:

1. **Binary classification** — decide whether a hostname/domain is DGA-generated (`isDGA = 1`) or legitimate (`isDGA = 0`).
2. **Subclass classification**
    - If the sample is DGA, identify its malware family (`gameoverdga`, `cryptolocker`, `newgoz`, `nivdort`).

    - If the sample is legitimate, decide between the `Alexa` and `nonAlexa` subsets.

Why CNN and LSTM?

Model	Intuition
CNN	Character-level convolutions detect short, highly-informative n-grams—e.g. the high-entropy trigrams often produced by DGAs. Global-max pooling then distils the most salient features into a fixed-length vector.
LSTM	An LSTM cell maintains two states: the hidden state (h_t) and the cell state (c_t). Input, forget, and output gates learn to keep or discard information so that long-range dependencies (e.g. repeated prefixes/suffixes or global entropy) are preserved. This sequential memory can capture stylistic quirks unique to individual malware families.

---

2 Dataset & Pre-processing

- **Size:** 160 000 rows (exactly 80 000 DGA, 80 000 legitimate).

- **Features:** `domain`, `host`, `isDGA`, `subclass`.

- **Cleaning:** lower-case, strip extra dots, drop 2 null/duplicate rows.

Tokenisation workflow

1. Remove dots and split `host` into characters.

2. **Padding/Truncation:**
   - Average host length 23; maximum 70.

   - We standardised to **30 characters** (`max_len = 30`): pad with `0` or trim excess.

3. **Integer encoding:** `Tokenizer(char_level=True)` → integer ids starting at 1; `0` reserved for padding.

---

3 Model Architectures & Hyper-parameters

Layer	**Binary CNN**	**Binary LSTM**
Embedding	`VOCAB × 50`, input 30	same
Conv1D / LSTM	`Conv1D(128, kernel 5, ReLU)`	`LSTM(128)`
Pool / Dropout	Global-max → `Dropout(0.5)`	`Dropout(0.5)` → Dense 64 → `Dropout(0.5)`
Dense	64 (ReLU)	64 (ReLU)
Output	Sigmoid	Sigmoid
Epochs	**10**	**5**
Batch	128	256

Equal wall-clock budget.
CNN trains 2× faster per epoch; we therefore used 10 epochs for CNN vs. 5 for LSTM so each model trains for roughly **5 minutes**.

Subclass models

- **DGA-CNN:** same as binary CNN but soft-max over 4 classes.

- **Legit-CNN:** lighter `Conv1D(64)` + Dense 32, soft-max 2.

- **DGA-LSTM / Legit-LSTM:** analogous LSTM layers with 128 and 64 units respectively.

- All subclass models use `epochs = 10`, `batch = 128`.

Full hyper-parameters

```
optimizer = Adam(lr = 1e-3)        embedding_dim = 50
dropout   = 0.50                    kernel_size   = 5
```

---

4 Training & Evaluation Protocol

- **Split:** 80 / 20 stratified train-test.
- **Metrics:** Accuracy, Precision, Recall, **F1** for binary; Accuracy for multi-class plus class-wise confusion matrices.
- **Early attempts:** We evaluated a generic character-CNN snippet suggested by ChatGPT (baseline F1 0.97). Our tuned models out-performed it, so we removed the baseline to avoid redundancy.

---

5 Results

5.1 Binary DGA Detection

Model	Accuracy	Precision	Recall	**F1**
CNN (10 × 128)	0.991 6	0.991 0	0.992 3	**0.991 6**
LSTM (5 × 256)	0.980 9	0.979 4	0.981 9	0.980 7

Next, we started training the CNN model using 10 epochs and a batch size of 128. We achieved 99.16% F1 score for binary classification using CNN, which is an impressive performance that we achieved after tuning the hyperparameters and trying multiple padding approaches. On the other hand, we achieved 98.07% F1 score for our LSTM model.

	Pred Legit	**Pred DGA**
Legit	15 836	166
DGA	137	15 857

5.2 Subclass Prediction

Task	CNN Acc.	LSTM Acc.
DGA-family (4-way)	87.73 %	**88.55 %**
Legitimate (2-way)	62.73 %	**63.33 %**

Then, we trained the models CNN and LSTM to perform multiclass classification to predict the subclass of each data point. Here, we set a stable number of epoch (10 epochs) and a stable batch size (128). For DGA subclasses, our

CNN model achieved 87.73% accuracy compared to 88.55% accuracy for our LSTM model, which shows good performance for both models. However, our models struggled with legitimate subclasses prediction with our CNN model achieving 62.73% accuracy compared to the 63.33% accuracy by LSTM model. Our models underperforming could be due to a number of reasons such as few epochs or imbalanced dataset. We need to perform manual error analysis to identify the issue and try to optimize our hyperparameters to achieve a better performance.

---

6 Discussion

- **Local vs. sequential bias** — For coarse binary separation, local n-gram cues dominate, hence CNN reaches > 99 % F1. For finer family discrimination the LSTM's long-range memory proves marginally superior.

- **Legitimate subclass bottleneck** — Both models saturate near 63 %. Reasons: class imbalance, subtle lexical overlap between Alexa and nonAlexa, and perhaps insufficient epochs. TF-IDF features or longer training could help.

- **Baseline comparison** — A quick ChatGPT-generated CNN achieved respectable performance (F1 0.97) but still lagged behind our tuned architecture by ~ 2 percentage points, validating the benefit of manual hyper-parameter search and padding optimisation.

---

7 Conclusion

This lab demonstrates that lightweight character-level deep networks can deliver state-of-the-art results for DGA detection:

- **Binary task:** A 3-layer CNN hit **99.16 % F1** with only 10 epochs.

- **Subclass task:** LSTM edged out CNN on malware-family prediction (88.55 % vs 87.73 %).

- **Future work:** address legitimate subclass imbalance, integrate entropy or TF-IDF features, and experiment with Transformers or CNN–LSTM hybrids.

Early, accurate DGA detection strengthens network defences; the presented pipeline therefore offers a fast, high-precision baseline for real-world deployment.

(Full notebook, confusion matrices, and graphs are included in the accompanying repository.)