Using AI for Unicode Spoof Detection

Name: Om Doiphode

University: Veermata Jijabai Technological Institute

Course: Bachelor of Technology (B.Tech.)

Branch: Computer Engineering

Physical Location: Mumbai, Maharashtra, India

Timezone: Indian Standard Time (UTC+5.30)

Resume

GitHub

LinkedIn

Table of Contents

Overview

Attackers can exploit homoglyphs, or visually similar characters, to perform social engineering attacks or evade spam and plagiarism detectors. Unicode contains over 200 writing systems (scripts). Many of these scripts, such as Latin (in western Europe) and Cyrillic (in eastern Europe), contain characters that are look-alikes. This project aims to develop an AI tool to automatically detect confusable characters and ultimately improve Unicode security mechanisms.

Goals

1. Build an AI model that evaluates the similarity between two characters. Use existing fonts as training data.
2. Run the model over Unicode code points to generate confusability data.
3. Submit the new confusability data to the UCD.
4. Stretch: Automate the system to run with the push of a button in the cloud.

In-depth Description

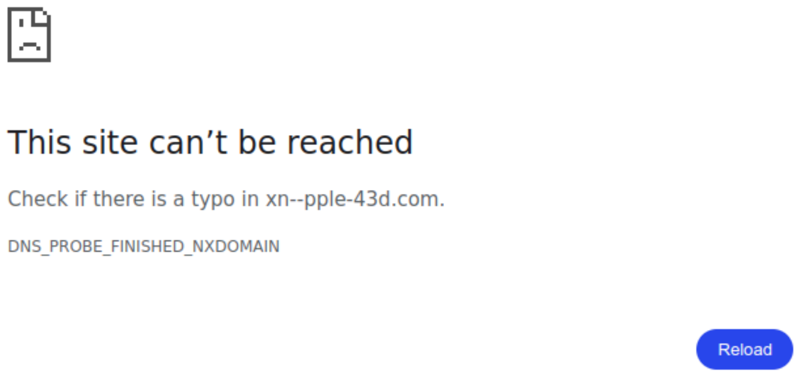
In orthography and typography, a homoglyph is one of two or more characters with shapes that appear identical or very similar. Characters that look the same can be swapped in strings to make strings that look similar. These strings can trick users into clicking on fake websites that seem real or avoid spam and plagiarism detectors. While people have known about the security risks of these look-alike characters for a long time, it's become a bigger issue as Unicode has become the main way text is processed. Unlike ASCII, which has only 128 characters, the newest Unicode standard has over 140 thousand characters. A real-world example of such a risk is a Punycode attack.

Punycode encode

Punycode is an encoding system that converts internationalized domain names (IDNs) containing non-ASCII characters into an ASCII-compatible format, ensuring compatibility with the **domain name system (DNS)**. By transforming characters like **é, a, or** 日 into an **ASCII-friendly** representation, Punycode allows the use of accented letters and non-Latin scripts in domain names. This enhances internet accessibility while preserving interoperability with existing infrastructure.

Homograph attacks exploit the visual similarities between Unicode and ASCII characters to create deceptive domains that closely resemble legitimate ones. For instance, an attacker might replace the Latin "a" and "p" in "apple.com" with their Cyrillic counterparts, making the domain appear nearly identical to the real one. While the fraudulent domain might look like "apple.com" to the naked eye, it uses different character encodings. Since modern browsers often display Punycode domains in their Unicode form, unsuspecting users may not notice the subtle differences, making it easier for attackers to lure them into phishing scams or malware-infested websites.

Consider the domain name **apple.com.** Here, the first "a" is replaced with the Cyrillic "a" (U+0430). If we try to access this, we get the following:



This site can’t be reached

Check if there is a typo in xn--pple-43d.com.

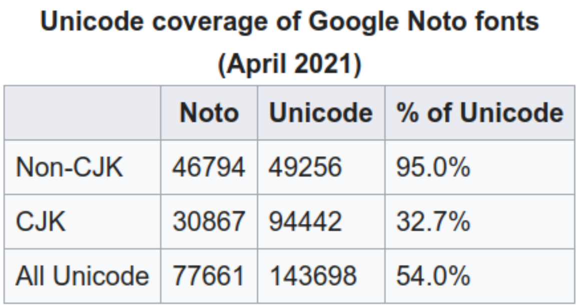
DNS_PROBE_FINISHED_NXDOMAIN

As can be seen here, the browser (Chrome in this instance) converts the entered URL into ASCII characters using a Punycode encoding scheme. Since **xn-apple-43d.com** is not registered in the DNS, it doesn’t redirect us to any site. But an attacker can register this domain in the DNS, and the victim might click on this domain of apple and would be redirected to the malicious site.

Dataset Preparation

As of **Unicode version 16.0**, there are **155,063 characters** with code points, covering 168 modern and historical scripts, as well as multiple symbol sets. The fonts available currently don’t support all the characters of Unicode. Google Noto fonts have the most extensive coverage of Unicode, covering 54% of Unicode. This means that only 54% of the total Unicode characters can be rendered by the Noto font. Thus, for the codepoint to be renderable by a font, it must have the following:

- The character must be on the list of the font's supported characters
- The character must not be replaced by any placeholder character by a font if there is no renderable font for it.



Unicode coverage of Google Noto fonts (April 2021)

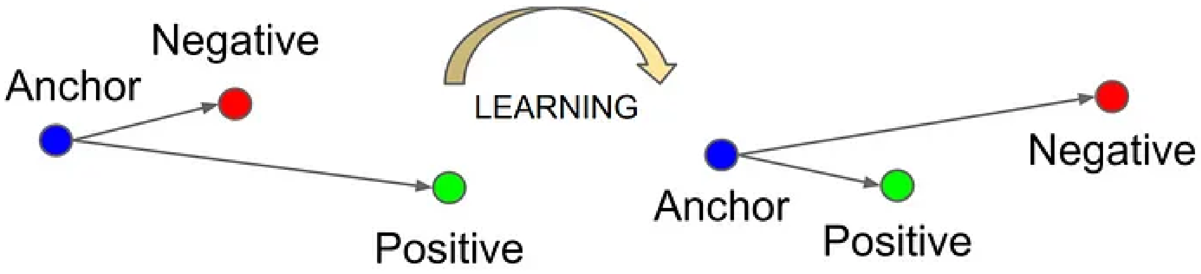
	Noto	Unicode	% of Unicode
Non-CJK	46794	49256	95.0%
CJK	30867	94442	32.7%
All Unicode	77661	143698	54.0%

For preparing the dataset, a collection of system (Ubuntu, Windows) fonts and Noto fonts will be used. To increase the diversity of the dataset, **data augmentation** will be used. These augmentations would account for real-world variations in text appearance. Some of the techniques are **blurring (Gaussian**), **noise (Gaussian)**, **distortion**, and **geometric transformations like rotation, flipping, warping, brightness and contrast changes**, **shadowing effects, and variable font weights.** This will ensure a broad representation of characters in our dataset, which is essential for training a robust CNN model.

Deep Learning Approach

We don't have explicit labels for the generated dataset. So, weak labels will be generated as follows:

1. Different codepoints are different characters.
2. Different Codepoints from Different Unicode Scripts Are Initially Considered Different. Since Unicode is organized into scripts (e.g., Latin, Greek, Cyrillic), characters from separate scripts are initially assumed to be unrelated.



Triplet Loss

Instead of training a classification model, we learn an embedding function that outputs relevant latent variables. Neural network architectures are good candidates for the embedding functions because they can approximate highly non-linear functions. To train this embedding function, triplet loss is used. The Triplet Loss function consists of three data points: an anchor data point, a positive data point from the same class as the anchor, and a negative data point from a different class. The objective of Triplet Loss is to minimize the distance between the anchor and positive embeddings while maximizing the distance between the anchor and negative embeddings.

For computing triplet loss, generally, euclidean distance is used, but here, **(1 - cosine similarity)** will be used. **EfficientNet-B7** will be chosen for embedding learning due to its efficiency and effectiveness in image classification tasks. EfficientNet-B7 will be used for this project as it is a larger and more powerful model compared to smaller variants like Efficient-B0.

Use a clustering algorithm to separate learned embeddings into equivalence classes:

A clustering algorithm will be used to separate learned embeddings into equivalence classes aimed at organizing characters based on their visual similarities. Since the number of clusters is unknown, using the KMeans clustering algorithm won't be beneficial. The **k-means** algorithm also assumes that the clusters are spherical and equally sized, which may not always be the case.

For our task, we can utilize hierarchical clustering, **DBSCAN (Density-Based Spatial Clustering of Applications with noise)**, or variants of DBSCAN like **HD DBSCAN (Hierarchical DBSCAN)**. Every algorithm has its pros and cons, and we will choose the one that gives a better IOU.

Evaluate the model using the Unicode consortium confusable database as ground truth:

The Unicode Consortium confusable database serves as a valuable resource for validating the model's performance and assessing its ability to identify homoglyphs accurately. By comparing the predicted equivalence classes generated by the model with the known homoglyphs in the database, we can evaluate the model's precision, recall, and overall effectiveness in detecting visually similar characters.

Timeline and Research Plan

Period	Work Expected
Community Bonding Period (8th May - 31st May)	
8th May to 31st May	● Getting feedback from the mentors about the submitted proposal. ● Discuss the high-level details of the project (goals, priority and deliverables, meeting schedule) with mentors. ● After this, create a general structure of the workflow of the project and adjust the timeline and research plan accordingly.
Coding (1st June - 25th August)	
Week 1-2 (1st June - 14th June)	● Set up the development environment and GitHub repository. ● Prepare the dataset for training the CNN model. ● Write a blog on Medium documenting the progress.
Week 3-4 (15th June - 28th June)	● Prepare the training script for the EfficientNet-B7 model. ● Train the model and use MLFlow or CometML to track the model training and conduct different experiments. ● Develop the clustering algorithm for grouping homoglyphs. ● Write a blog on Medium documenting the training of the model.
Week 5-6 (29th June - 12th July)	● Look for alternative clustering algorithms. ● Fine-tune the model further for better metrics. ● Write a blog on Medium.
Week 7 (13th July - 19th July)	● Note down the performance of the

	different models trained and tested. ● Consolidate all the blogs written on Medium till 12th July and prepare a mid-term report.
Week 8 - 9 (20th July - 2nd August)	● Utilize the fine-tuned model to identify errors in the Unicode Consortium Confusables.txt database. ● Update the database with corrected pairings and missing homoglyphs. ● Use the fine-tuned EfficientNetB7 model to predict and validate new homoglyphs not present in the database. ● Write a Medium article documenting the progress.
Week 10-11 (3rd August - 16th August)	Conduct thorough testing and debugging to ensure the reliability of the adapted model.
Week 12-13 (17th August - 25th August)	● Finalize the project implementation and results after discussion with the mentor. ● Prepare the final report. ● Use this time for any final adjustments, optimizations, or additional tasks.
Week 14 (26th August - 1st September)	Submit the final report on the GSoC portal after reviewing it from the mentor.

About Me

I am currently a third-year undergraduate student at Veermata Jijabai Institute of Technology, Mumbai, Maharashtra, India, pursuing a Bachelor of Technology in Computer Engineering and will graduate in July 2025. I have experience in Python, C, C++, React, and Node. I was also a GSoC contributor in 2024 at Data Retriever (NumFOCUS). I am interested in this project since it provides me an opportunity to work on homoglyph detection, which will prove useful in preventing phishing attacks in real world systems.

Open Source Contributions

1. **GSoC'24 Contributor at DeepForest**
2. **DeepForest (Pre GSoC'24)**
    - **Fixed deprecation warning when running main.deepforest()**
      https://github.com/weecology/DeepForest/pull/375
    - **Fixed Syntax Warning**
      https://github.com/weecology/DeepForest/pull/565
    - **Added doc string for save_dir in evaluate function**
      https://github.com/weecology/DeepForest/pull/596
    - **Fixed error related to tile assignment**
      https://github.com/weecology/DeepForest/pull/599
    - **Updated Configuration.md file**
      https://github.com/weecology/DeepForest/pull/623
    - **Updated environment.yml file**
      https://github.com/weecology/DeepForest/pull/624
    - **Fixed deprecation warning**
      https://github.com/weecology/DeepForest/pull/629
    - **Gradio application for Getting Started Tutorial**
      https://github.com/weecology/DeepForest/pull/637
3. **Ivy**
    - **Add logical_and method to PyTorch Frontend**
      https://github.com/unifyai/ivy/pull/13114 (merged)
    - **Add bitwise_left_shift method to PyTorch Frontend**
      https://github.com/unifyai/ivy/pull/13120 (merged)
4. **MindsDB**
   Implemented insert, update, and delete queries for the carrier service table
   https://github.com/mindsdb/mindsdb/pull/8028
5. **Openvino ToolKit (NNCF)**
   Replaced Runtime Error
   https://github.com/openvinotoolkit/nncf/pull/2412

Other Commitments

During my summer holidays (June to July), I will have no major commitments and can dedicate 6 hours daily to the project. I plan to split this time into 7-8 hours for core development and 1-2 hours for learning and research. From August to September, I will allocate 3-4 hours daily to ensure steady progress.

However, my commitment goes beyond fixed hours - I will make the most of this opportunity by working as much as needed to deliver the best possible results.

References

1. Punycode encoding scheme
2. The Punycode Problem: Protecting Your Inbox from Deceptive Domains
3. Weaponizing Unicodes with Deep Learning - Identifying Homoglyphs with Weakly Labeled Data
4. Triplet Loss