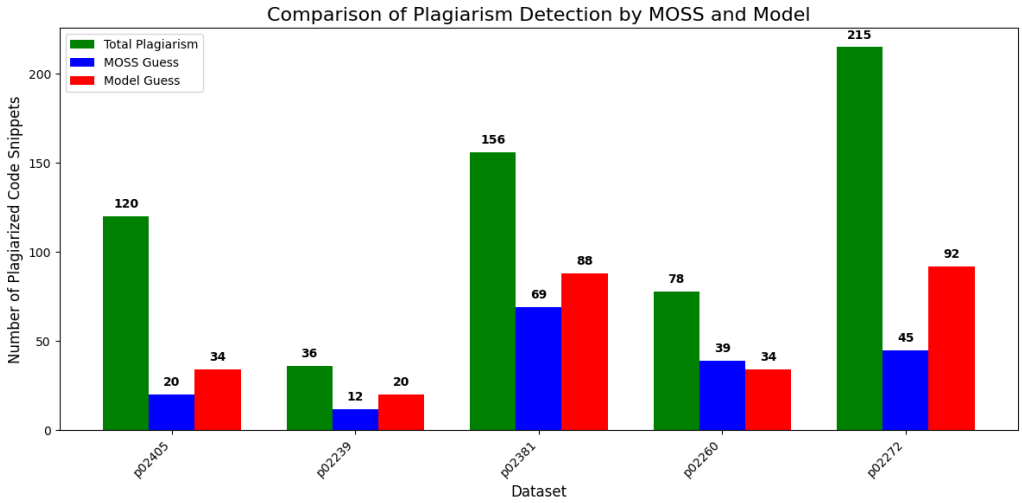
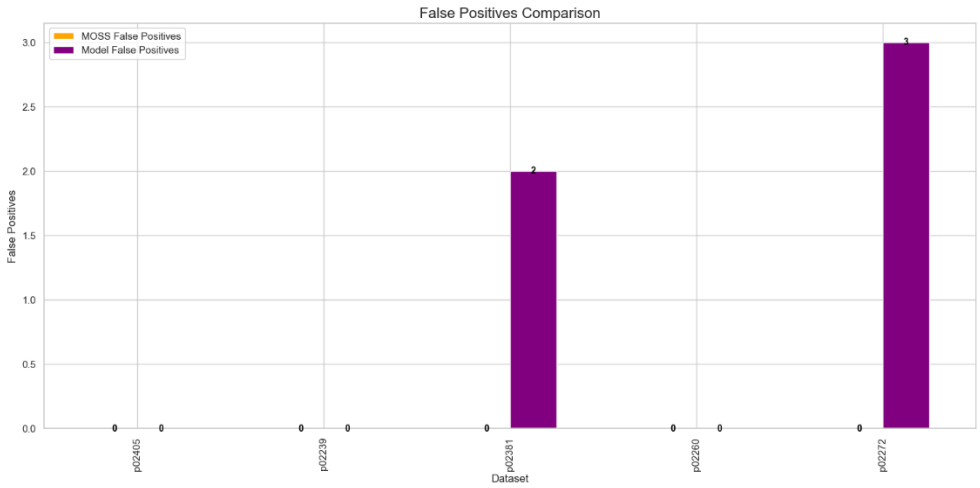
Benchmark for Models

Overview: IBM provides online datasets to help train “algorithmic innovations in AI” with over 14 million code samples [1]. Microsoft has utilized such datasets to train their own NLP models for comprehending code on a semantic level, such as their codeBERT model [2]. Our own model utilizes codeBERT, so we felt it appropriate to utilize this data.

One of the contained datasets is named “Project_CodeNet_Python800”. This dataset contains folders that house around 300 python files on average that range in line count from 3 all the way to 150 (between folders).

The data in its raw form is unlabelled. Our team manually labelled around 20 folders to establish our plagiarism-detection comparison in the graph below. This comparison is between our model and MOSS [3], the current standard for plagiarism detection.





Per our results, our model provides less leniency for code comparison compared to MOSS, making it more likely to punish plagiarism. However, it also has demonstrated a propensity for false positives that MOSS does not have, albeit insignificant.

The folders used possess 303-307 code snippets each and the average lines of code in each folder ranges anywhere between 3 to 40 lines to provide variety for comparison.

Methodology for labelling:

The team randomly selected 3-6 code snippets from roughly 20 folders, plagiarized each of them, and re-entered tampered snippets into the dataset. It then utilized a third-party MOSS-like algorithm (tokenization and fingerprinting comparison) to apply similarity scores for all pairings within the folder.

From here, it became a matter of philosophy. Our team had the following beliefs:

1) False positives should be minimized
2) Similarity is not a direct indicator of plagiarism. However, for the sake of training, if a file pair had a higher similarity score than the files we injected, it was much more likely that it was plagiarized.
3) Injected files cannot be modified too heavily, as this produces an issue of classifying ownership akin to the Ship of Theseus (i.e., at what point is ownership passed to the adversary after modifying enough of the code). Our plagiarism methods were restricted to minimal structural change or prompting ChatGPT to improve code while changing the names of functions encountered (something we imagined future students will do frequently).

This philosophy led us to label the dataset as follows:

1. Compare the maximum plagiarism scores of each injected code file. Take the highest score of these maximums. This gives a score which a pair of code snippets must be *particularly* egregious to be reach this score. Note: different plagiarism methods seemed to claim this spot without anyone dominating alone (i.e., sometimes GPT came up as the 'winner' while other times a manual method did).
2. Any plagiarism score between this max injection score and the middle max injection score had its maximum score adjusted by the difference between them divided by the number of files between them.

$$AdjustedMaxScore = CurrentMax + \left(\frac{MaxInjection - MiddleMaxInjection}{FileCountBetween}\right)$$

The reasoning was that we believed files that were above the middle max injection score were still suspicious and should receive greater spotlight. Here, we also abided by our philosophy to reduce false positives in two ways. By dividing by the file count between, we were easing punishment on files the more

were found between, because that meant there was a higher likelihood it was a coincidence. Also, we did not punish any files with max scores below the middle max injection score and the min max injection score (even though we knew they were somehow doing worse than sloppily plagiarized files). We chalked this up to the possibility the problem being solved by the code submissions was simple enough in nature to prevent as much individualistic expression as well as wanting to be less punitive in the model we trained.

3. Any file with an adjust_max_score above the max injected score was labelled as plagiarized. The idea is that it doesn't matter how many files a code snippet performs low on, if it has a particularly high score with any single file, it has a possibility to be plagiarized. In other words, plagiarism does not present itself via aggregate but by a case-by-case basis (although aggregates like mean similarity can provide much context).

Based on this labelling, we ran a MOSS algorithm against our model for comparison. If the maximum plagiarism score for a code snippet came below the threshold for a dataset, a code snippet would be labelled as 0. Otherwise, it would be labelled as 1. This is what informed our metrics above. Our team acknowledges that it is heavily skewed by multiple factors, but we came to understand as this project evolved that plagiarism is a much more fickle/less concrete matter to assess than we originally understood. It is very hard to obtain a plagiarism-labelled dataset without manually procuring one over a long duration (longer than this class had scope for). Even with such a dataset, one must ask, how can plagiarism be assessed?

References

[1] https://developer.ibm.com/exchanges/data/all/project-codenet/

[2] http://www.mysmu.edu/faculty/lxjiang/papers/apsec23interpretCodeBERT.pdf

[3] https://theory.stanford.edu/~aiken/moss/