Qeios

Peer Review

Review of: "An Empirical Study on Automatically Detecting AI-Generated Source Code: How Far Are We?"

Mikhail Evtikhiev[1]

1. Independent researcher

Review:

While the paper has many merits, the assessment approach chosen by the authors is, unfortunately, insufficient to analyse the problem posed. Therefore, I suggest **major revision** for this paper.

Overview: In this paper, the authors do a critical review of the existing methods and tools used to identify AI-generated code. They observe that these tools have very low accuracy (of circa 50% or less) and propose three alternatives: 1. using LLM such as ChatGPT to identify AI-generated code, 2. using ML classifiers over static code metrics, 3. using ML classifiers over code embeddings. They find that the ML classifiers over code embeddings, with code embeddings built from the code AST, perform the best, reaching an accuracy of circa 80%.

Major concerns

1. Choice of training and evaluation dataset. The datasets the authors choose are very reasonable in general, though they are quite old; choosing newer datasets or benchmarks like LiveCodeBench could have addressed some of the data contamination concerns. However, these datasets may not be fully suitable for the task at hand for the following reasons:
    1. Both HumanEval-X and MBPP contain LeetCode-style tasks, which are not representative of real-world code. While by itself this is no reason to discard this data (as identifying AI-generated LeetCode-style code could be helpful for e.g. identifying cheating at interviews or home assignments), it would make sense to assess AI code identifiers in two different setups: “real code” (which the authors address through usage of the CodeSearchNet (CSN) dataset) and “leet code”. The authors do that in some of the cases, which brings us to another issue:

 doi.org/10.32388/PN9K1L

2. The test dataset size is in many cases inadequate to assess model performance. As the authors note themselves in the paper (page 10), 100% accuracy of GPTSniffer on one particular occasion could be related to the small number of test samples (33 samples). The scores of models assessed on datasets like that will have very wide error bars, which may prohibit making meaningful statements about relative model performance.
    1. To address the two points above, I suggest extending the CSN dataset (at least its test part, so it will have at least circa 300 examples for each case considered independently) and reporting the scores on CSN and on HumanEval-X+MBPP separately. While this may be hard due to the enormous number of experiments carried out by the authors, I strongly believe doing this in at least several key setups (a couple of classifiers the authors created, GPTSniffer, LLM-as-a-judge, and one or two classifiers they review) would greatly improve the paper's findings. I also advise the authors to estimate the error bars for at least the most important tables they provide. To do so, they could either refer to a paper by Miller (https://arxiv.org/pdf/2411.00640) or use methods like bootstrapping; the former approach is likely to be faster and simpler to execute.
3. Table 1, with the description of the datasets, is a bit unclear: while the number of samples in the authors' MBPP dataset is roughly twice the size of the original MBPP dataset (which makes sense as the authors provide two solutions for each problem: human and AI), for the other datasets it is way less than twice the dataset size. While for HumanEval-X this could be explained by the fact that each language version only has 164 problems (which is unclear from the body of the paper), for CSN it seems that filtering for code syntax errors has removed approximately 50% of the dataset (if the paper's statement that the authors have chosen 400 instances each for Python and Java is correct). Why could this be the case?
4. The authors conduct many experiments (e.g. they create 48 different datasets of code later used in many different experimental setups), which greatly increases the odds of false positive observations. While in most cases the authors draw their conclusions from the aggregated scores, so this risk is mitigated, I advise the authors to acknowledge it directly and verify (by using e.g. Bonferroni correction) that the results they use to draw their conclusions are statistically significant.
5. The concern about score significance and error bars is also relevant for the cosine similarity between various code sources the authors report. While this observation could have significant value, the absence of statistical evaluation makes it somewhat unfounded.

2. My second major concern is related to the choice of metrics and baselines for the task. In this paper's setup, the code classifiers essentially classify the code into two classes of equal size, which causes the following concerns for me:
    1. In such a setup, even a classifier that outputs random scores (or always says the code is e.g. AI-generated) will have 50% accuracy. Therefore, using accuracy (as well as TPR, TNR, and F1-score) may be somewhat misleading, as a classifier with 50% accuracy (the score which is better than the reported scores of many classifiers) is as good as a coin toss.
        1. To address that, I suggest using metrics like the Brier score or ECE (cf. Spiess et al., https://arxiv.org/pdf/2402.02047), and reporting a “random” classifier as an artificial baseline.
    2. In many cases (e.g. GPT-2 output detector for ChatGPT and Gemini, table 4), the scores seem to contradict each other. For example, if there is an equal number of human and AI-generated code (as per the described setup), then, I believe, TPR=1 and TNR=0 (or vice versa) should yield an accuracy of 50%. However, this is often not the case.
        1. I advise the authors to re-check the reported scores, and kindly ask them to either fix possible errors or explain this (seeming) discrepancy.

Minor concerns / clarification requests:

1. What is the difference between ChatGPT and GPT-4? It is not clear from the paper.
    1. A minor comment on the model choice: one can argue the choice is not too representative as the authors only choose one OSS model and do not consider other models like Codestral, DeepSeek-coder, Qwen-2.5-coder, or CodeLLama. Given the extensive work the authors did and the focus on the AI classifiers and not on the relative model performance, I would argue this is not a significant problem.
2. In the last sentence of the second paragraph on page 4, you say <...> , *we randomly sampled 400 (confidence level 95%, margin of error 5%) data instances each for Python and Java.* What do confidence level and margin of error mean in this case?
3. The authors conduct the experiments for the LLMs at both T=0 and the recommended LLM temperature. While I appreciate the authors’ desire to obtain reproducible results, I would argue that using an LLM at T=0 makes little sense as it is not the mode users will use it in (cf. section 3.3 in the paper by Miller, https://arxiv.org/pdf/2411.00640;). While this does not constitute a problem by itself

(especially given that the authors mainly rely on the results for code generated at the non-zero temperature), the corresponding pieces of the paper could be safely omitted.

    1. This also means that the experiments with the CSN extension I suggest should not be done at T=0 temperature, hopefully saving some of the authors' time and resources.
    2. To address the reproducibility issue, it is possible to generate many solutions at the given temperature. Given the size of the study, I definitely do not ask to do that for this paper.

4. Did you do the hyperparameter tuning reported on page 7 on the dev portion of the dataset? This is worth a two-word clarification for the reviewer (and reader) peace of mind.
5. You mention that sometimes classifiers you study classify all provided code as human-written code. Do you have any speculation for why this is so?
6. The data you report for GPTSniffer show it performs quite badly on Java code, which seems to contradict the original paper. Do you have any explanation for why this is the case?
7. The classifiers you propose (both LLM-as-a-classifier and metrics/embedding-based ones) seem to have less bias towards classifying all the code as human-generated. Do you have any ideas why they have less of this bias?

Technical comment: In some cases (e.g., the first sentence on page 7), the table links are broken.

Declarations

Potential competing interests: No potential competing interests to declare.