HumanVsGenAI: AI Code Detection in Educational Programming Tasks using DetectCodeGPT

Aagat Pokhrel, Chandra Raskoti, Chong Shen, Iftekharul Islam

Introduction

- LLMs like ChatGPT & DeepSeek are widely used in coding
- Blurring boundaries raises new challenges for instructors
    - detection, fairness, and trust issues

Related Work

- Three approaches to code detection:
  - Probability-based: AIGCode [1]
  - Supervised learning: GPTSniffer [2]
  - Perturbation-based: **DetectCodeGPT** [3]

- **Gap:** Very few studies in educational contexts

[1] Yang, Xianjun, et al. "**Zero-shot detection of machine-generated codes.**" arXiv preprint arXiv:2310.05103 (2023).
[2] Nguyen, Phuong T., et al. "**GPTSniffer: A CodeBERT-based classifier to detect source code written by ChatGPT.**" Journal of Systems and Software 214 (2024).
[3] Shi, Yuling, et al. "**Between lines of code: Unraveling the distinct patterns of machine and human programmers.**" arXiv preprint arXiv:2401.06461 (2024).

Research Question

How effective is DetectCodeGPT in detecting AI-generated solutions across educational programming tasks?

Dataset Collection

Dataset	Solution Source	Type of Students	No. of Problems	AI Models Used
A	GitHub	Intermediate / Experienced	35	ChatGPT-4, Claude 3.7 Sonnet, DeepSeek-V3
B	Univ. of Hamburg	CS1/CS2 Students	21	Same

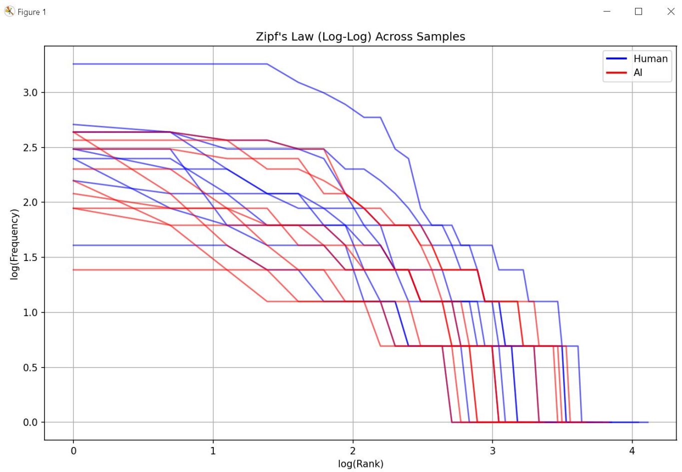
Dataset Exploration

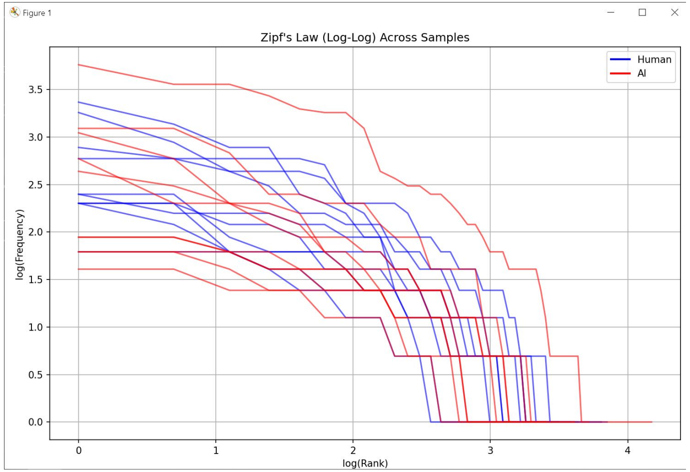
Two factors important to us:

1. **Lexical Diversity:** How varied is the vocabulary? We used extensive Language Tree Parser
2. **Naturalness:** How predictable are the patterns? We used several different LLM models

Lexical Diversity

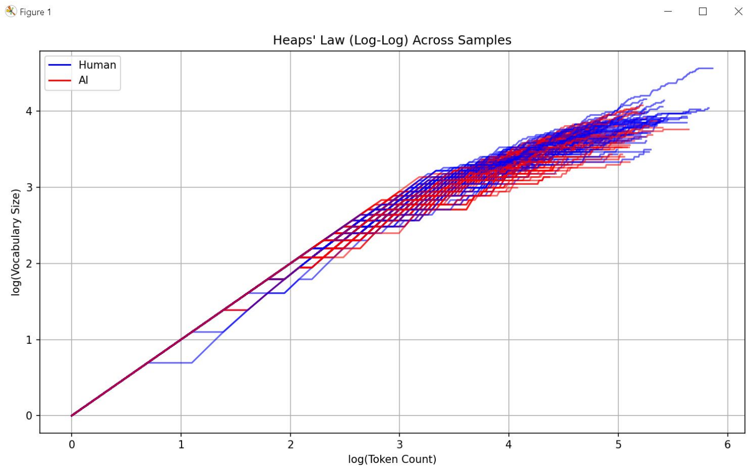
Zipf's Law : represents token frequencies follow a power-law distribution

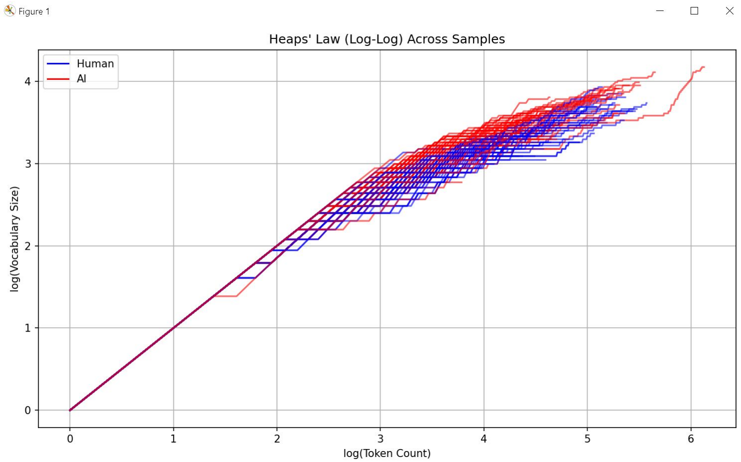




Lexical Diversity

Heap's Law : how the vocabulary grows as the total tokens increases in a corpus





Naturalness

Log Likelihood:

- Log probability the model assigns to the actual next token in a sequence.
- Reflects how correct the model is in its prediction

Log Rank:

- Rank of a token is its position in the model's sorted prediction list
- For whole code, it is the average log rank.

Naturalness

Category	Logrank (AI) Mean	Logrank (AI) Std	Logrank (Human) Mean	Logrank (Human) Std
comment	8.29	0.36	8.15	0.80
identifier	6.11	0.75	5.81	0.66
keyword	9.35	0.62	9.27	0.63
literal	4.84	0.56	4.69	0.87
operator	6.19	0.32	6.31	0.31
whitespace	5.43	0.27	5.18	0.35

Category	Logrank (AI) Mean	Logrank (AI) Std	Logrank (Human) Mean	Logrank (Human) Std
Comment	8.35	0.38	7.48	0.36
Identifier	5.96	0.65	6.06	0.74
Keyword	9.39	0.61	9.43	0.46
Literal	5.26	0.85	4.83	0.39
Operator	6.26	0.27	6.10	0.31
Whitespace	5.50	0.30	5.08	0.38

Note: This analysis was done on a smaller model and our small dataset samples

Methodology

- **Analysis :** Machine seem to write natural code
- **Strategy:** Devise method of perturbation to affect the naturalness
- **Measure :** Compare naturalness after perturbation
- **Evaluation :** Performance evaluation of the overall system
- **Research Problem :** How to define the naturalness score, perturbation process and evaluation metric.

Perturbation Strategy

1. **Space Insertion**
   - Inserted randomly

2. **Newline Insertion**
   - Similar to space insertion.

We randomly choose type perturbation to the code snippet x.

Naturalness Measure

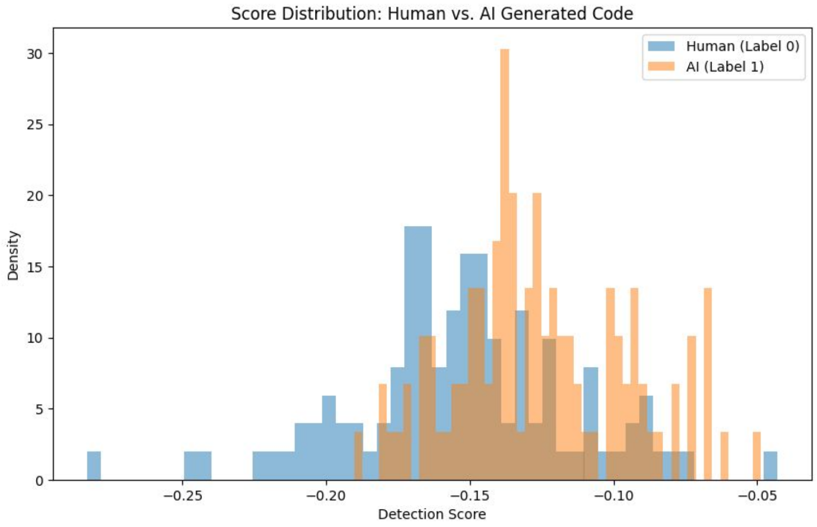
We adopt the Normalized Perturbed LogRank (NPR) score to capture the naturalness. The NPR score is formally defined as:

$$\mathbf{NPR}\left(x, p_\theta, q\right) \triangleq \frac{\mathbb{E}_{\tilde{x} \sim q(\cdot|x)} \log r_\theta\left(\tilde{x}\right)}{\log r_\theta(x)},$$

Experiment

- Perturbations limited to 10 permutations
- For newline insertion, α to be 0.5 (50%) and λ spaces=3
- For space insertion, β to 0.5 and λ newlines=2
- CodeGen (350M params, 2.3B params) and Llama (7B params) used

Results



Score Distribution: Human vs. AI Generated Code
Human (Label 0)
AI (Label 1)
Density
Detection Score
0
5
10
15
20
25
30
−0.25
−0.20
−0.15
−0.10
−0.05

Example1

[Trial] A simple example, showing human code in the left vs ai code in the right

```
def factorial (n) :
    if n == 0:
        return  1
    else :
        return n  *  factorial(n-1)


Human code - Score: 0.82
```

```
def factorial(n):
    return 1 if n == 0 else n * factorial(n - 1)

Machine code - Score: 0.9540449312935378
```

Example2

A complex assignment problem in our dataset

```
def editDistance(pre, after) :
    if len(pre)  >  len(after ):
        pre, after =  after, pre

    result = range (len(pre) + 1)

    for i, b in  enumerate (after):
        tmp = [i+1]
        for j, a in enumerate (pre):

            if a !=  b:

                tmp.append (1 + min((result[j], result[j + 1], tmp[-1])))
            else:
                tmp.append (result[j])

        result =  tmp
    return result[-1]
Human code - Score: 0.88
```

```
def editDistance(a, b):
    if len(a) > len(b):
        a, b = b, a
    total = range(len(a) + 1)
    for i_b, s_b in enumerate(b):
        tmpdist = [i_b+1]
        for i_a, s_a in enumerate(a):
            if s_a == s_b:
                tmpdist.append(total[i_a])
            else:
                tmpdist.append(1 + min((total[i_a], total[i_a + 1], tmpdist[-1])))
        total = tmpdist
    return total[-1]
Machine code - Score: 0.9249366846400272
```

Evaluation

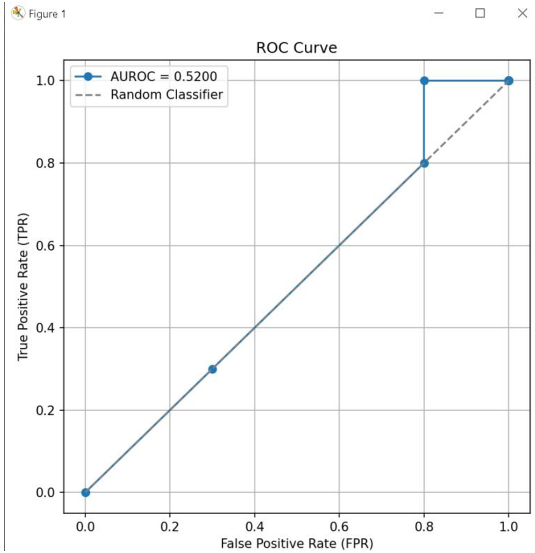
AUROC quantifies a model's ability to distinguish between positive and negative classes across all threshold levels.

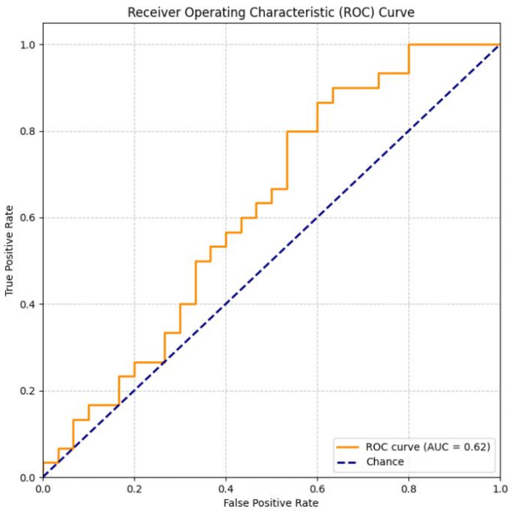
$$\mathrm{AUROC} = \int_0^1 \mathrm{TPR}(t)\, dt,$$

where t denotes varying threshold values

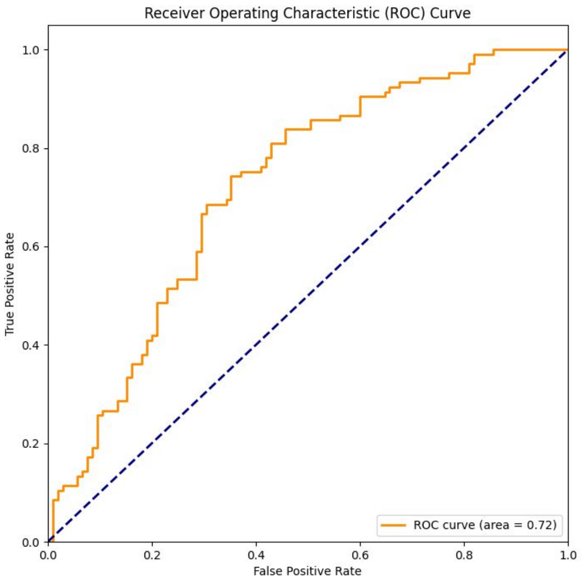
Evaluation

Small model (Codegen 350M params) in left vs Large Model (Llama 1.3B params)

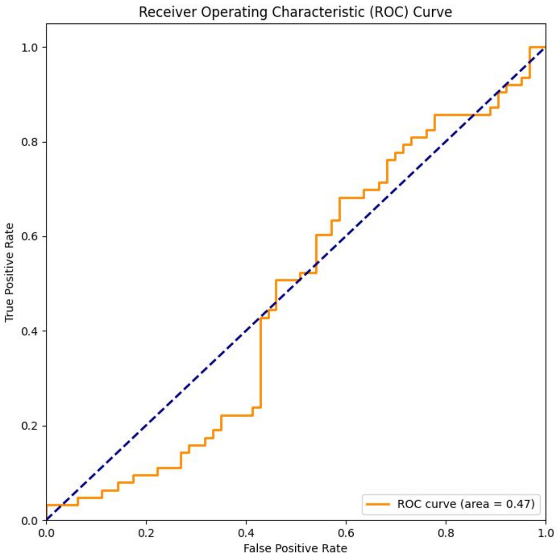




Evaluation: Codegen 3.7B params

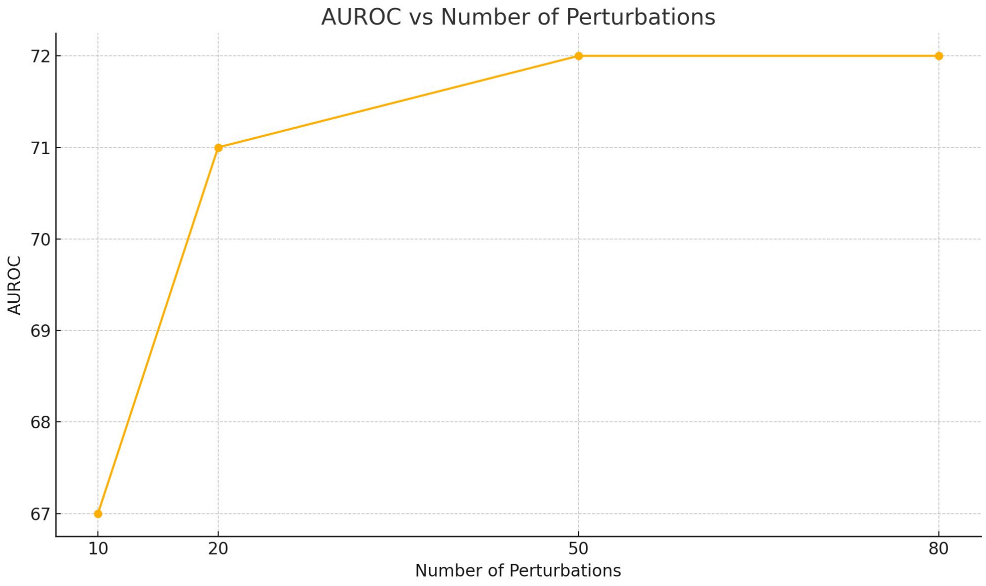


GitHub



Student: Univ. of Hamburg

AUROC at different perturbations



Limitations

- Rely entirely on the formatting features(spaces, newlines, and indentation) to tell the difference between machine and human
- Ignore the logic, control flow and API usage
- As a result, very neatly formatted human code can be mistakenly flagged as machine-written.

Limitations

- Our experiments so far only covered models up to 7 billion parameters, such as CodeLlama-7B.
- Much larger LLMs—13 B, 30 B, 70 B and beyond—have become commonplace.
- The experiment shows that as the model become larger, spacing and token patterns become smoother, which makes the perturbations less effective.

Limitations

- The code we collected from public repositories may itself contain AI-generated content.
- This potential label noise could blur the distinction between our positive (human) and negative (machine) examples, thereby undermining the validity of our evaluation.

Future Work

1. **Incorporate semantic signals** (e.g. API usage) alongside formatting perturbations to better distinguish well-formatted human code.
2. **Extend evaluation to larger LLMs** (13B, 30B, 70B+) and re-tune our perturbation strengths and thresholds so the method remains effective as models continue to grow.
3. **Use datasets of more programming languages**(e.g. Java, C++, JavaScript) into our evaluation and perturbation framework to ensure DetectCodeGPT generalizes across different syntax and style conventions.

Conclusion

- Our experimental results fell short of expectations: even the best-performing solution struggled to distinguish AI-generated code
- This highlights that whitespace-only perturbations are not sufficient enough.
- Going forward, we will refine our approach by adding semantic and structural features, and evaluating on larger models and more varied datasets to boost detection accuracy.