Between Lines of Code: Unraveling the Distinct Patterns of Machine and Human Programmers

Aagat Pokhrel, Chandra Raskoti, Chong Sen , Iftekharul Islam

The Challenge

- AI-generated code increasingly indistinguishable from human code
- Blurring boundaries raise new challenges for software engineering

- **Why Detection Matters:**
  - **Team development:** Who to consult for bugs?
  - **Security:** Different review standards needed
  - **Management:** Accurate productivity measurement

Why Previous Methods Fall Short

- Text detection methods like DetectGPT:
  - Analyze log probability curvature
  - Measure likelihood difference after perturbation
  - Designed for natural language patterns

- Why they fail with code:
  - Programming languages follow strict syntax rules
  - Random perturbation often breaks functionality

Research Approach

- Three key dimensions analyzed:
    1. **Lexical Diversity:** How varied is the vocabulary? (Token frequency, Syntax distribution, Zipf's & Heaps' laws)
    2. **Conciseness:** How compact is the code? (Token count, line numbers)
    3. **Naturalness:** How predictable are the patterns? (Likelihood and rank metrics)

- **Goal**: Find reliable patterns that distinguish machine from human code

Data and Experimental Setup

- Human Code Source:
  - 10,000 Python functions from CodeSearchNet
  - Diverse range of real-world GitHub projects
  - Function signatures with comments used as prompts

- Machine Code Generation:
  - CodeLlama (7B parameters)
  - Incoder (1.3B)
  - Phi-1 (1.3B)
  - StarCoder (3B)
  - WizardCoder (3B)
  - CodeGen2 (3.7B)

- Generation settings:
  - T=0.2: Standard, predictable code
  - T=1.0: Creative, diverse solutions

Findings from Initial Analysis

Results and analysis

Finding 1

- Machine-authored code focuses more on: **exception handling**, **object-oriented principles**
- Reason: machine focuses on avoiding mistakes and following standard practices
- Almost no differences in token frequency

Results and analysis

Finding 2

Machine-authored code tends to use:

- Fewer identifiers(names for variables, functions, or classes)
- More literals(numbers or strings)
- More comments

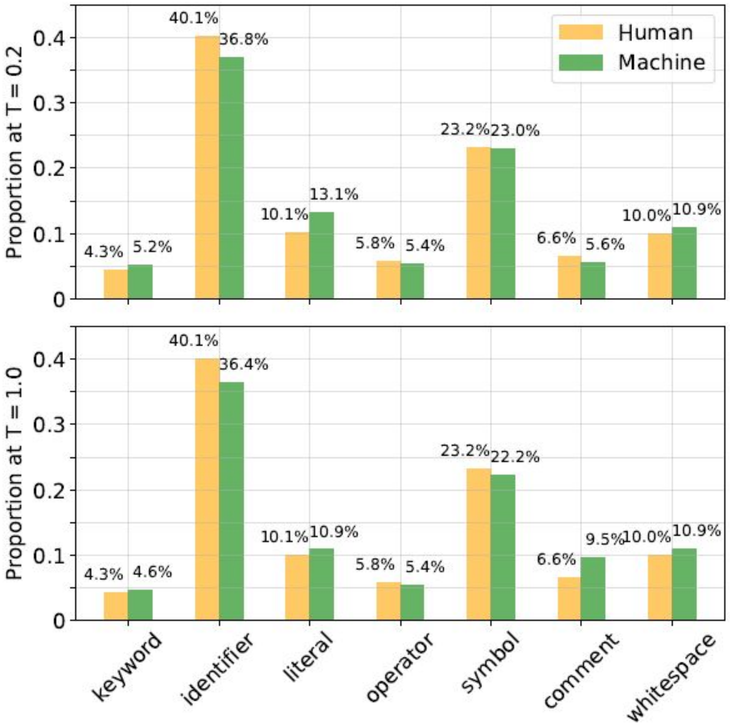
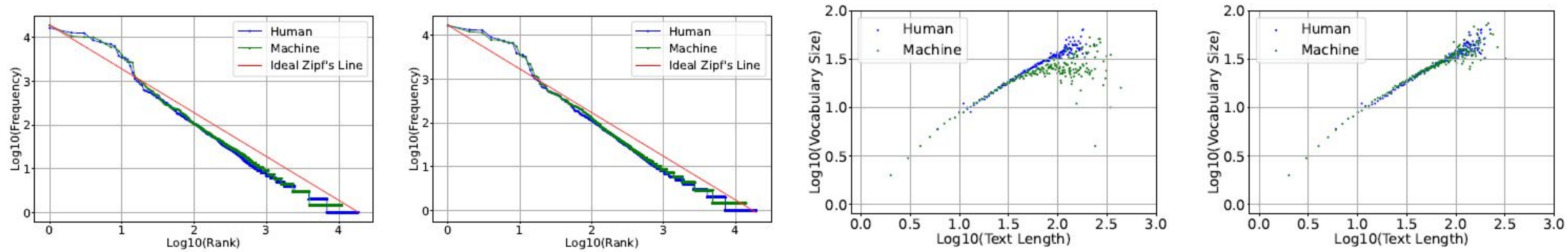


Figure 1: Syntax element distribution of the code corpus

Results and analysis

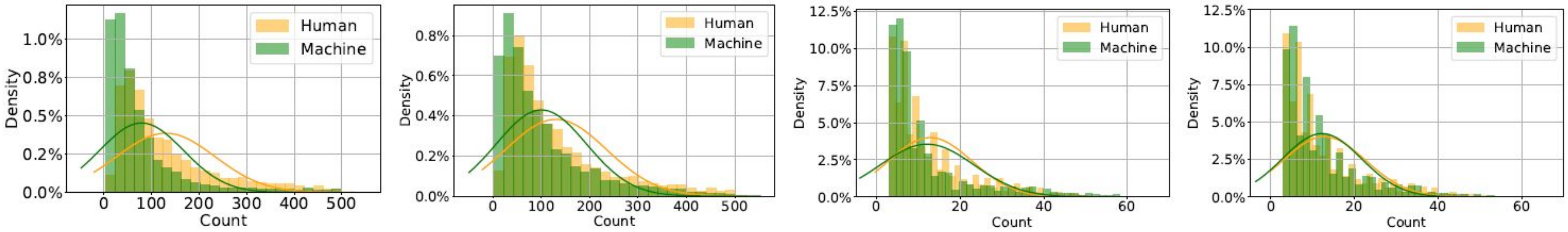
Finding 3

- Machines demonstrate a preference for a limited spectrum of frequently-used tokens.
- Human code exhibits a richer diversity in token selection.



(a) Zipf's Law ($T = 0.2$) (b) Zipf's Law ($T = 1.0$) (c) Heaps' Law ($T = 0.2$) (d) Heaps' Law ($T = 1.0$)

Figure 2: Comparison of Zipf's and Heaps' laws on machine- and human-authored code



(a) Number of tokens ($T = 0.2$) (b) Number of tokens ($T = 1.0$) (c) Number of lines ($T = 0.2$) (d) Number of lines ($T = 1.0$)

Figure 3: Distribution of code length for machine- and human-authored code

Results and analysis

Finding 4

- Machines tend to write more concise code
- Reason: Training objective is to produce more efficient code
- Human programmers tend to write longer code
- Reason: personal stylistic preferences and sometimes more detailed explanations.

Results and analysis

Finding 5

- Machine-authored code exhibits higher "naturalness" than human-authored code
- probability distribution of tokens in machine code fits well with what the model learned during training

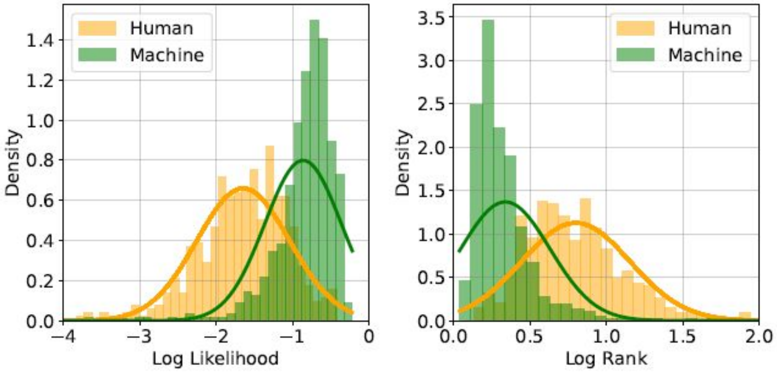


Figure 4: Distribution of naturalness scores

The DetectCodeGPT method, and Evaluation Setup

Idea of method

- A classification task : which predicts whether or not a given code snippet x is produced by resource model
- Instead of perturbing arbitrary tokens, we focus on perturbing those stylistic tokens of a code
- Machine seem to write natural code, we need to disturb this naturalness
- The key problems are how to define the naturalness score and how to design the perturbation process

Naturalness Score

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

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

Perturbation Strategy

1. **Space Insertion**
   - Inserted randomly
   - α to be 0.5 (50%) and λspaces=3

2. **Newline Insertion**
   - Similar to space insertion.
   - β to 0.5 and λnewlines=2

We randomly choose type perturbation to the code snippet x.

Evaluation

Formally, given a set of true positive rates (TPR) and false positive rates (FPR) across different thresholds, the AUROC can be represented as:

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

where t denotes varying threshold values

Results, Performance Comparisons, and Conclusions

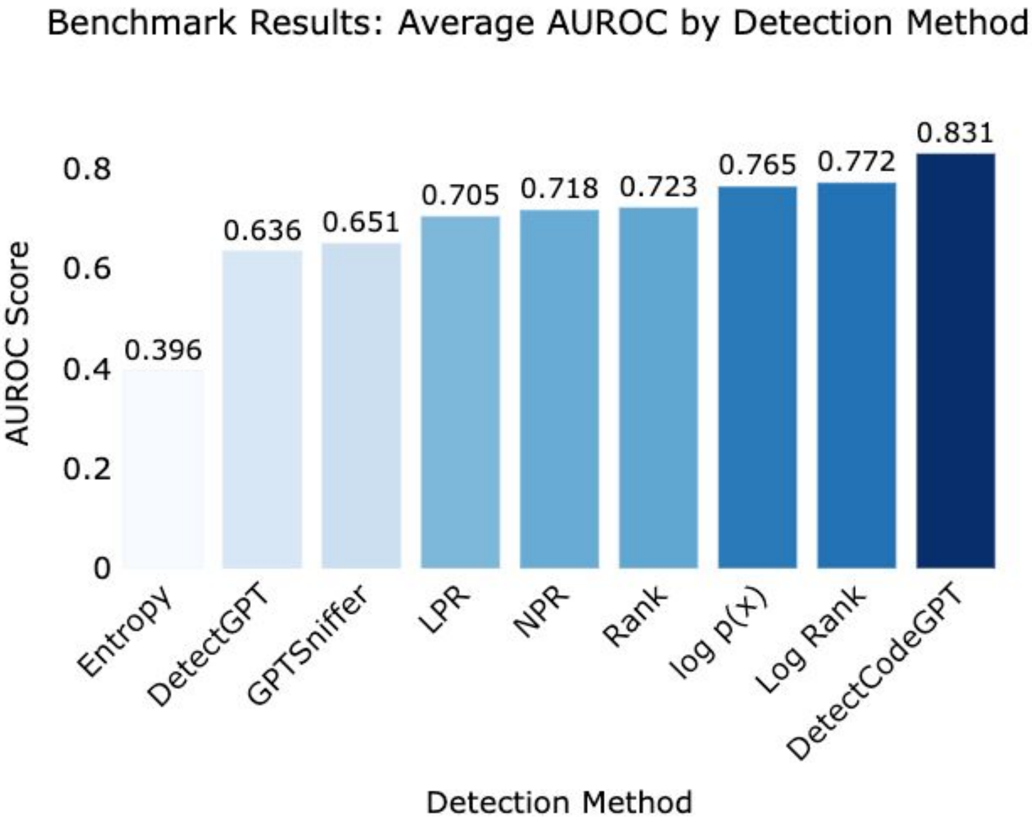
Overall Performance

83.08%

Overall AUROC

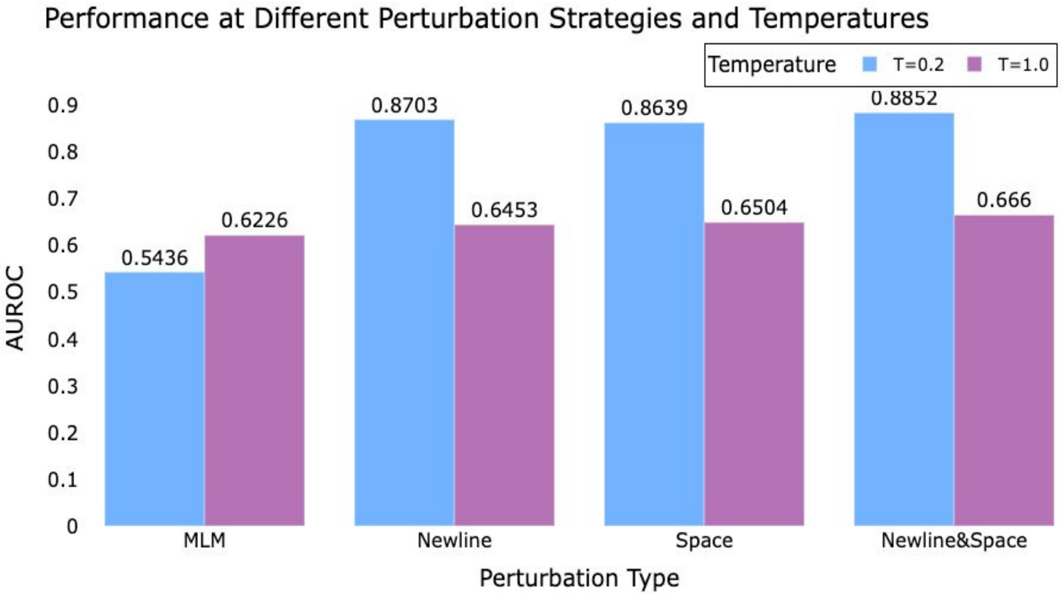
+7.6%

Improvement Over Best Baseline



Robustness Analysis

- **Robust Performance:** DetectCodeGPT maintains effectiveness with **different level of perturbations**
- **Enhanced Detection** : Combining both perturbations **improves accuracy** across different randomness levels (temperature settings).

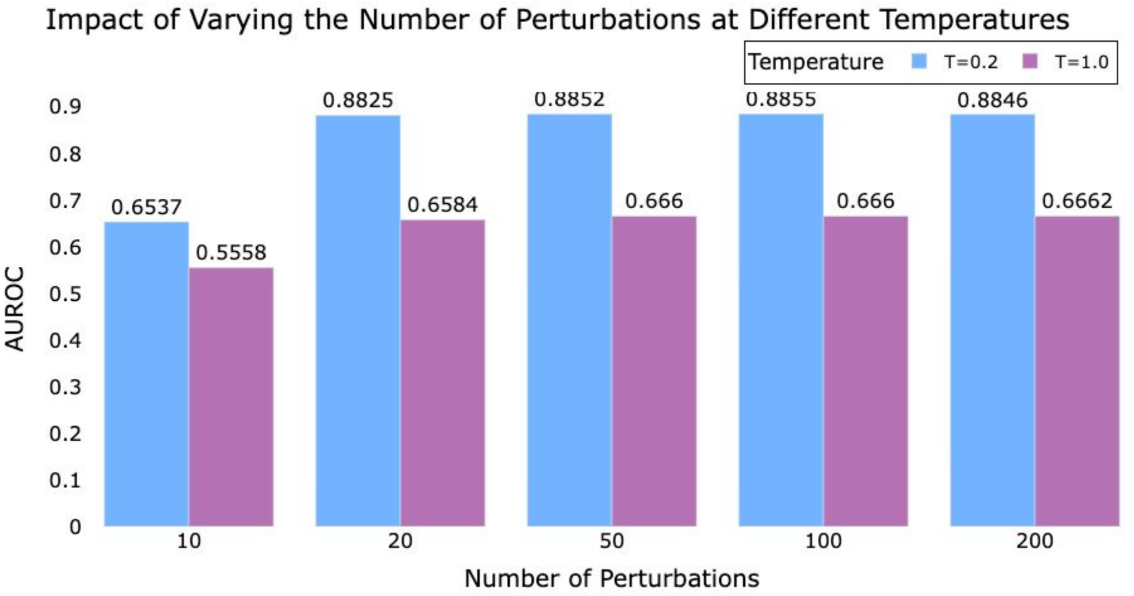


*MLP is the traditional perturbation technique used in DetectGPT

Impact of Number of Perturbations

- **Improvement with More Perturbations** : Detection performance increases as the number of perturbations rises.

- **Good enough at small Perturbations** : Further gains beyond **20 perturbations** are minimal, indicating **efficiency with a small number of perturbations**.



Limitations

- **Limited LLM Scope** : Current analysis focuses only on models up to **7B parameters**, limiting generalizability to complex codes.
- **Python-Only Evaluation** : The study is restricted to **Python code**, and effectiveness on other languages is not fully explored
- Limited Perturbations: Space and Newline

Future Research

- **Expanding LLM Coverage** : Incorporate larger and more diverse **LLMs** to enhance robustness.
- **Multi-Language Generalization** : Extend analysis to other languages
- **High-Randomness Code** : Experiment with AI-generated code with higher variability
- **Advanced Perturbation Strategies** : Explore code style-based techniques beyond simple whitespace insertion

Conclusions

- **Key Insights** : Machine-generated code is **more concise, natural, and structured**
- **Proposed Method** : Introduced **DetectCodeGPT**, leveraging a **perturbation strategy** to detect AI-generated code.
- **Effectiveness** : Experiments confirm **high performance** in distinguishing human vs. machine generated code.
- **Impact** : Helps **preserve authorship and integrity** in coding.