The Underlying Differences Between AI and Human Coding

The rapid development of artificial intelligence (AI) has led to its widespread adoption in software development, including in areas like code generation and competitive programming. AI tools like OpenAI's Codex and ChatGPT, as well as specialized models such as GitHub Copilot, have demonstrated impressive abilities to generate code from natural language descriptions. This paper seeks to directly address the debate surrounding the quality, adaptability, and maintainability of AI-generated code versus human-created solutions by conducting a comparative analysis across a variety of competitive programming problems.

Previous research, such as DeepMind's 2022 evaluation of AlphaCode, explored AI's performance in competitive programming, highlighting its efficiency and scalability in solving algorithmic challenges. However, the study also noted limitations in AI's adaptability and ability to handle problems requiring creativity or broader context.(ar5iv).

While AlphaCode's study highlights AI's algorithmic efficiency, this research focuses on broader metrics like readability, commenting, variable naming, and modularity—crucial for long-term code quality. Instead of just evaluating performance, it compares AI and human solutions in real-world software practices, offering a broader view of AI-generated code's strengths and weaknesses across various coding scenarios.

Understanding how AI-generated code compares to human-generated code is critical, not only for improving AI's capabilities but also for safeguarding the integrity of coding education and assessments. This comparison sheds light on key differences that impact efficiency, readability, and adaptability, providing insight into how AI could either support or hinder human learning and creativity.

In this experiment, I compared human and AI (GPT 4o) coding solutions across a range of competitive programming problems, assessing various metrics to explore the fundamental differences between human and AI approaches. The results provide crucial insights into the distinct characteristics of AI-generated code and establish a foundation for developing AI detection tools in the coding domain.

Example Analysis of Human and AI Solutions

To get a preliminary glance into some things that may differ between Human and AI code, solutions to a single problem were compiled and analyzed..

```cpp
# Human Solution
#include <bits/stdc++.h>
using namespace std;

/**
 * Calculate the sum of all elements in arr, represented by the binary mask, and
 * take modulo mod.
 */
int unmask(int mask, const vector<int> &arr, int mod) {
    int current_sum = 0;
    for (int bit = 0; bit < arr.size(); bit++) {
        if ((mask >> bit & 1) == 1) {
            current_sum += arr[bit];
            current_sum %= mod;
        }
    }
    return current_sum;
}

int main() {
    int n, m;
    cin >> n >> m;

    // split the input array into two parts
    vector<int> left_arr(n + 1) / 2);
    vector<int> right_arr(n / 2);
    for (int &i : left_arr) { cin >> i; }
    for (int &i : right_arr) { cin >> i; }

    // stores the sums of all combinations from the left_arr modulo m
    set<int> left_sums;
    for (int mask = 0; mask < (1 << left_arr.size()); mask++) {
        left_sums.insert(unmask(mask, left_arr, m));
    }

    // the best value from all combinations of left_arr
    int best = *left_sums.rbegin();
    for (int mask = 0; mask < (1 << right_arr.size()); mask++) {
        int current_sum = unmask(mask, right_arr, m);
        /*
         * a possible new maximum value is the sum of current_sum and the
         * largest value below m - current_sum from the combinations in left_arr
         */
        best = max(best, *prev(left_sums.lower_bound(m - current_sum)) + current_sum);
    }

    cout << best << endl;
}
```

```cpp
# AI Solution
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main() {
    int n, m;
    cin >> n >> m;
    vector<int> a(n);

    for (int i = 0; n; ++i) {
        cin >> a[i];
    }

    // Divide the array into two halves
    int half = n / 2;
    vector<int> left, right;

    // Generate all possible subsets sums modulo m for the left half
    for (int i = 0; (1 << half); ++i) {
        int sum = 0;
        for (int j = 0; j < half; ++j) {
            if (i & (1 << j)) {
                sum = (sum + a[j]) % m;
            }
        }
        left.push_back(sum);
    }

    // Generate all possible subsets sums modulo m for the right half
    for (int i = 0; i < (1 << (n - half)); ++i) {
        int sum = 0;
        for (int j = 0; j < n - half; ++j) {
            if (i & (1 << j)) {
                sum = (sum + a[half + j]) % m;
            }
        }
        right.push_back(sum);
    }

    // Sort the right half for binary search
    sort(right.begin(), right.end());

    int max_value = 0;

    // For each element in left, find the best complement in right
    for (int x : left) {
        // We want to maximize (x + y) % m
        // So we find the largest y such that x + y < m
        int complement = m - x - 1;
        auto it = upper_bound(right.begin(), right.end(), complement);
        if (it != right.begin()) {
            --it;
            max_value = max(max_value, (x + *it) % m);
        }
        // Try also combining with the smallest element
        max_value = max(max_value, (x + right[0]) % m);
    }

    cout << max_value << endl;

    return 0;
}
```

Figure 1: Example of Human and AI solutions to a competitive coding problem

Key Differences Between Human and AI Solutions

- Cyclomatic Complexity: The human solution introduces slightly higher complexity by using a modular function (unmask), which helps with reusability, while the AI solution handles everything in main() for lower complexity but at the cost of readability.
- Lines of Code (LOC): Both solutions have similar LOC (43 for human, 44 for AI), but the human solution reduces repetition by reusing

code in a function, while the AI solution has some redundancy within main().

- Commenting: The human solution contains both single-line and multiline comments, which enhance readability and explain the logic. In contrast, the AI solution has no comments, making it harder to understand without prior knowledge of the code.
- Variable Names: The human solution uses more descriptive variable names (left_arr, right_arr, current_sum), improving clarity. The AI solution uses more generic names (left, right, a), which can reduce readability.
- Modularity: The human solution separates logic into a reusable function, making it more maintainable. The AI solution is monolithic, which simplifies the flow but makes it harder to extend.

While these initial observations provide valuable insights into differences between AI and human coding, the following methodology aims to verify whether these trends hold true consistently across a larger dataset of 200 coding problems.

Methodology Overview: Comparative Analysis of Human and AI Solutions

A total of 200 competitive coding problems were selected using the competitive coding resource of USACO Guide. These problems were drawn from platforms such as USACO, CSES, and Codeforces. The sampling method involved stratified random sampling, where eight strata were created based on two dimensions: problem difficulty (easy, normal, hard, very hard) and algorithm type (common and uncommon). This ensured that each stratum was represented equally, providing a balanced dataset for analysis. A simple random sample (SRS) of 25 questions was performed for each stratum, ensuring balanced representation across difficulty and algorithm type.

For each selected problem, both human and AI-generated (GPT 4o) solutions were obtained in C++. Human solutions were extracted from the "View Solution" tab on USACO Guide, while AI solutions were generated using the prompt: "Give me a solution to this competitive coding problem in C++." These solutions were then saved in txt files, with the format [human/computer]*[difficulty]* [common/uncommon].txt, where each solution was separated by `/n New Solution /n`.

Each txt file was then analyzed using a custom-built program to extract key metrics, namely:

- **Cyclomatic Complexity**: A measure of the logical complexity of the solution, based on the number of independent paths through the code.
- **Lines of Code (LOC)** and **Source Lines of Code (SLOC)**: The total number of lines and the number of executable code lines, respectively.
- **Number of Functions**: The number of distinct functions defined in the code.
- **Single-Line Comments** and **Multiline Comments**: Metrics for the different styles of comments used.
- **Total Comments**: The sum of all comments.
- **Blank Lines**: The number of blank lines in the solution.
- **Number of Includes**: The number of external libraries included.
- **Variables (Split Into Different Types)**: The number of variables, analyzed through the diversity of data types used.
- **Compile Rate**: The percentage of solutions that successfully compiled.

To find more details about the custom program that the solutions were run through, as well as all of the gathered solutions, please visit the GitHub repository.

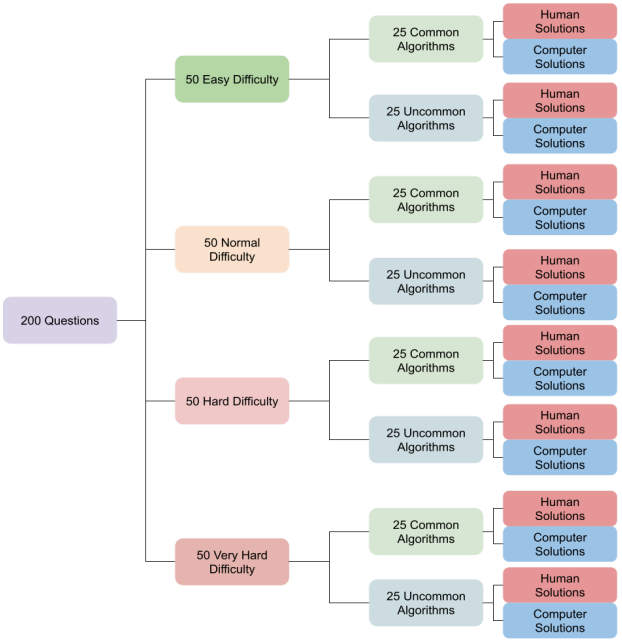

Figure 1: Flowchart of 200 competitive coding problems. Split first by difficulty, then by common/uncommon computer algorithms.

Key Observations into Human and AI Coding Approaches

For a complete insight on the information gathered, visit this spreadsheet.

1. LOC/SLOC Percentage: Human Solutions Exhibit Higher SLOC

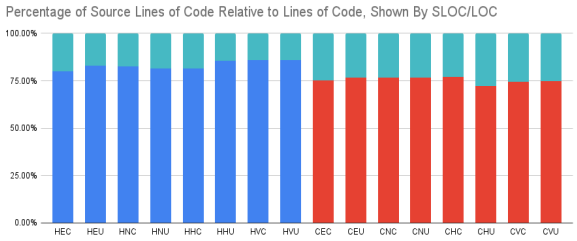

Figure 2: Shows the SLOC/LOC percentage for each subsection. Abbreviations are as follows: H-human or C-computer, E-easy, N-normal, H-hard, V-very hard, C-common, U-uncommon. Blue and red bars show the percentage of source lines, while turquoise shows the rest(non-source) lines.

One of the most pronounced differences observed between human and AI solutions is the significantly higher percentage of **source lines of code (SLOC) relative to total lines of code (LOC)** in human solutions. As it can be seen, AI solutions consistently exhibit SLOC to LOC percentages of above 80%, while human solutions are normally around 75%. This difference may be attributed to the following factors:

- **Commenting Practices**: Human solutions consistently contain more single-line comments than AI-generated code, contributing to higher LOC but not SLOC.
- **Functions and Blank Lines**: Human solutions feature a greater number of blank lines, enhancing readability, while AI solutions favor simplicity. Additionally, human coders tend to break problems down into more functions as the complexity of the problem increases, contributing to the higher SLOC observed in their solutions.

2. Consistency in AI Solutions: Lower Standard Deviation in Key Metrics

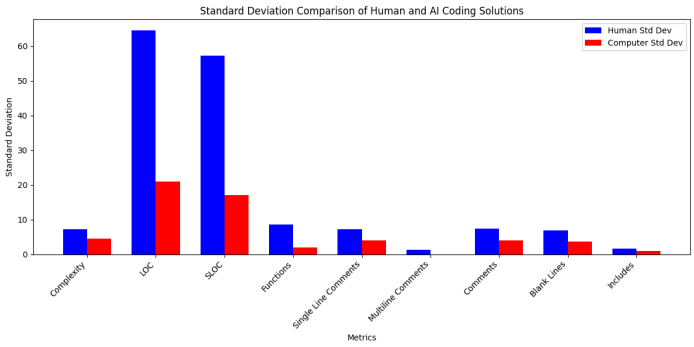

Figure 3: Standard deviations of the applicable metrics. Shows consistently greater human variance compared to AI solutions.

Another critical observation is the significantly lower standard deviation in AI-generated solutions compared to human solutions, indicating a much higher level of consistency in how AI approaches coding. This is evident across several key metrics, including lines of code (LOC), source lines of code (SLOC), and the number of functions. AI solutions demonstrate impressive consistency in the number of source lines of code (SLOC), generally falling between 40 and 55 SLOC across all difficulty levels. In contrast, human solutions exhibit a clear relationship between problem difficulty and SLOC, with the number of lines increasing as the complexity of the task rises, ranging from around 45 SLOC for easy solutions all the way to 110 SLOC to difficult questions . This underscores a key difference in problem-solving approaches: AI tends to maintain a streamlined and compact code structure irrespective of the difficulty, whereas humans are inclined to expand their code as the problems grow more complex, indicating a more iterative and exploratory process.

Additionally, the standard deviation in lines of code (LOC) for AI solutions is 20.90, compared to 64.46 for human solutions. This significant disparity reinforces the point made earlier, suggesting that while human coders vary considerably in their methods, AI solutions remain relatively uniform across different problems. Similarly, the deviation in SLOC for AI is 17.04, compared to 57.22 for humans. These figures illustrate how AI adheres to a more consistent approach in both structure and code length, regardless of the problem's complexity.

Moreover, in terms of the number of functions used, AI solutions exhibit a deviation of only 2.00, compared to 8.52 for humans. This further reinforces the idea that AI tends to follow a more uniform coding structure, while humans adapt their code more flexibly based on the problem, leading to greater variability.

To statistically validate these observations, I conducted an F-test for each metric to compare variances and assess whether the differences in standard deviations between human and AI solutions were statistically significant. The results indicated a p-value of less than 1×10^{-8} for all metrics analyzed, providing strong evidence that the variance in AI solutions is significantly lower than that of human solutions. This finding supports the conclusion that AI demonstrates a more consistent coding structure

across different problems, albeit with reduced adaptability compared to human solutions.

The reduced variability in AI solutions can be seen as both a strength and a limitation. On the one hand, it ensures that AI consistently produces concise, efficient code. On the other hand, this lack of variation may suggest that AI solutions are less adaptable to the specific nuances of different problems, unlike human coders, who demonstrate more diverse and flexible problem-solving approaches.

3. Commenting Patterns: Multiline Comments in Human Solutions

A key distinction lies in the use of **multiline comments**. While human coders frequently incorporate multiline comments to describe complex logic or explain sections of their code in greater detail, **AI-generated solutions never use multiline comments**. This lack of comprehensive documentation in AI code suggests that AI solutions, while often concise, may be less suited for collaborative or educational contexts where understanding the rationale behind code is crucial.

4. Variable Types: Greater Diversity in Human Solutions

Both human and AI solutions exhibit a reliance on basic variable types, such as `int`, with humans using a significantly larger quantity of variables overall. Human coders demonstrate not only a higher count of variables but also a greater diversity in variable types, employing more specialized data types such as `double`, `bool`, and `string` more frequently than AI. This suggests that humans tend to tailor their solutions more precisely to the given problem, while AI often defaults to simpler, more generalized data types. The considerable difference in total variable usage by humans highlights their more nuanced understanding of problem requirements, which can lead to more efficient and readable solutions. AI, on the other hand, exhibits more consistency across different problems but may lack the depth of human reasoning when selecting variable types.

Total Variable Usage by Humans and AI Over 200 Questions, Separated by Variable Type

	Solution Type			
Variable Type		Human	Computer	**Total**
	int	3449 (91.9%)	1813 (95.2%)	5262 (93%)
	double	46 (1.2%)	13 (0.7%)	59 (1%)
	bool	189 (5%)	51 (2.7%)	240 (4.2%)
	string	6 (0.2%)	0 (0%)	6 (0.1%)
	char	63 (1.7%)	26 (1.4%)	89 (1.6%)
	float	0 (0%)	1 (0.1%)	1 (0%)
	Total	3753 (100%)	1904 (100%)	**5657 (100%)**

Figure 4: Shows number of the various primitive variables in C++ (including string) utilized by Human and AI coding solutions. Note that these numbers are the sum of all of the 200 sample solutions.

5. Compile Rates: Occasional Failures in AI Solutions

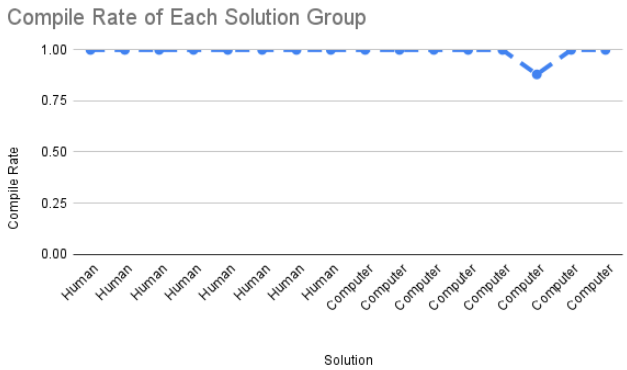

Figure 5: Shows the compile rate of each solution group as a decimal, 100% compile rate is 1.00. A 100% compile rate signifies that all solutions were able to be compiled.

In general, both human and AI solutions exhibited high compile rates. However, in one specific **strata**, three AI solutions failed to compile, while no human solutions encountered this issue. These isolated compile errors may indicate that AI, though highly competent in generating valid code, may occasionally produce solutions that fail to compile correctly. Although this is interesting, there is no significant difference between the compilation rate of human solutions and AI solutions that can let us easily differentiate between the two.

Conclusion: Key Differences Between Human and AI Competitive Coding Solutions

The comparative analysis between human and AI-generated competitive coding solutions reveals several fundamental differences, showcasing both the strengths and limitations of each approach. Human coders tend to produce more varied and diverse solutions, as reflected by higher standard deviations across key metrics such as LOC, SLOC, and the number of functions. This variability allows humans to adapt their solutions to the unique nuances and complexity of different problems, leading to greater modularity, flexibility, and more diverse use of variable types. Human coders also tend to include more detailed documentation, such as single-line and multiline comments, enhancing readability and facilitating collaboration in complex problem-solving scenarios.

In contrast, AI solutions exhibit remarkable consistency, as evidenced by significantly lower standard deviations in LOC, SLOC, and function usage. AI-generated code tends

to be concise and streamlined, with a predictable structure across difficulty levels. While this consistency ensures efficient and compact code, it also highlights a potential limitation—AI may lack the flexibility and adaptability that human coders display, especially in handling more intricate or varied problems.

The uniformity in AI solutions, coupled with their occasional failure to compile, suggests that while AI is capable of generating functional code, it may not fully understand the deeper logic behind complex tasks. Ultimately, while AI can generate concise and efficient code, human oversight remains crucial for handling complex, creative, and collaborative coding tasks. These results underscore the need for AI tools to evolve, not just in generating correct code, but also in improving flexibility, adaptability, and the ability to collaborate seamlessly with human developers.

Overall, these insights into the coding practices of humans and AI provide valuable implications for both educational and practical coding environments. AI can serve as an efficient tool for generating code, but human oversight and adaptability remain crucial for handling more complex, creative, and collaborative coding challenges.

Implications for AI Detection in Coding

The results of this experiment offer valuable insights into the distinct characteristics of human and AI-generated code, providing a foundation for developing tools that can detect AI involvement in coding tasks. The differences in commenting practices, function usage, LOC consistency, and variable types are key indicators that can be leveraged to distinguish between human and AI-generated solutions.

AI Detection for Educational Integrity
In educational settings, particularly in coding and computer science programs, it is essential to detect when learners rely too heavily on AI tools like ChatGPT. While AI-generated code can be correct and concise, it often lacks the depth of explanation, creativity, and iteration that are integral to the learning process. Detecting AI involvement ensures that students engage deeply with the material and develop the problem-solving skills necessary for real-world coding tasks.

Enhancing Coding Assessments
In competitive coding environments, the ability to identify AI-generated code is critical for maintaining fairness and ensuring that participants are evaluated based on their own coding abilities. By recognizing the trends in AI-generated solutions, educators and evaluators can develop more robust assessment criteria that differentiate between human and AI problem-solving approaches.

Ultimately, these findings underscore the importance of balancing AI assistance with personal development. While AI can be a powerful tool in coding, its overuse—particularly by those still learning—can undermine the acquisition of critical coding skills. By developing methods for detecting AI-generated solutions, we can ensure that AI serves as a complement to human learning rather than a substitute for it.

Limitations

While this study offers valuable insights, several limitations should be considered when interpreting the results.

First, it was challenging to gather a diverse set of competitive coding problems across various platforms, as many contests have limited available resources, both in terms of the number of questions and the accessibility of provided solutions. As a result, the study primarily relied on three major platforms—USACO, Codeforces (CF), and CSES—which may limit the generalizability of the findings to a broader range of coding problems. The problem selection is largely constrained to algorithms frequently tested in these environments, potentially omitting other problem types or structures that might reveal further distinctions between human and AI-generated solutions.

Another limitation lies in the sample of human solutions. All of the human-generated code was produced by the same group of individuals, which could introduce a consistent style or approach that may not fully represent the broader diversity of the coding community. This narrow sample might skew the results, as coding styles, problem-solving approaches, and practices vary widely among programmers, depending on experience levels, educational backgrounds, and personal preferences. The trends observed here may reflect this particular group's coding tendencies rather than capturing the full spectrum of human approaches to competitive programming.

Additionally, the experiment focused solely on C++ as the programming language for both human and AI solutions. While C++ is a widely-used language in competitive programming, the findings may differ if other languages, such as Python, Java, or JavaScript, were included in the

analysis. Different languages have distinct syntactic rules, best practices, and idiomatic conventions, which could impact how both humans and AI approach problem-solving in those contexts.

Finally, this study examines only AI solutions generated by a specific version of ChatGPT. As AI models continue to evolve, future iterations of ChatGPT or other models may exhibit different coding behaviors, potentially altering the comparisons made in this experiment. The performance and characteristics of AI solutions may change as these models become more advanced, and the results of this study should be viewed as a snapshot in time rather than definitive conclusions about AI's capabilities in competitive coding.

Despite these limitations, the findings provide a strong foundation for further investigation into the differences between human and AI coding approaches, with potential applications in AI detection and educational integrity in programming. Looking ahead, future research could explore how AI-generated solutions differ across different programming languages, such as Python or Java, and how newer versions of AI models might address some of the limitations observed in this study.

Credits

I would like to give a special thanks to Nick, David and the rest of the WashU NLP Group at Washington University in St. Louis, who have guided and advised me during the data collection, analysis, and the writing of this blog.