A Performance-Based Analysis of ChatGPT on Competitive Programming Challenges: A Replication

Daniel Trimble

September 11, 2025

Abstract

The integration of Large Language Models (LLMs) like ChatGPT into software development workflows has prompted significant discussion regarding their capabilities relative to human developers. While foundational studies have explored this relationship across a wide range of metrics, a need exists for focused, replicable performance benchmarks. This paper presents an agile replication of the work by Nascimento et al. (2023), concentrating specifically on the runtime and memory performance of ChatGPT-generated solutions on the LeetCode platform. I tested 30 problems of varying difficulty and compared the AI's "out-of-the-box" solutions against the platform's median developer benchmarks. The results indicate that ChatGPT **consistently** generates highly performant code in terms of runtime, with a median percentile of 87.04%, but demonstrates only average memory efficiency. A clear trend emerges where runtime performance, while still strong, degrades significantly **as problem complexity increases.** These findings contribute to a more nuanced understanding of the practical strengths and weaknesses of current-generation AI in performance-critical programming tasks.

1 Introduction

The proliferation of advanced AI, particularly LLMs, is rapidly changing the landscape of software engineering. Tools like ChatGPT are now widely used for code generation, debugging, and ideation. However, empirical validation of their performance against skilled human developers is still an emerging area of research. This study seeks to contribute to this area by providing a focused performance analysis.

Building on the comprehensive, multi-faceted methodology of Nascimento et al. (2023), this paper narrows its scope to answer three specific questions:

1. **Runtime Performance:** Does ChatGPT's generated code execute faster than 50% of human submissions on LeetCode?
2. **Memory Efficiency:** Is its code more memory-efficient than 50% of submissions?
3. **Consistency:** How does its performance vary across Easy, Medium, and Hard problems?

By concentrating on these objective, platform-provided metrics, I aim to deliver a clear and replicable benchmark of ChatGPT's capabilities.

2 Background: The Original Study

My research is a direct response to the foundational work "Comparing Software Developers with ChatGPT: An Empirical Investigation" (Nascimento et al., 2023). In their study, the authors conducted a broad comparison of ChatGPT-generated code against human solutions. Their methodology was notable for its holistic approach, analyzing not only performance (runtime) but also crucial non-functional requirements such as code quality, security vulnerabilities, and energy efficiency.

The key conclusion from their work was that while AI is a powerful tool, it cannot be seen as a direct replacement for human developers. They found that AI-generated solutions often failed on the critical non-functional requirements where human experience and oversight are essential. Their study advocated for a collaborative model of human-AI interaction in software development.

3 Methodology

To investigate my research questions, I adopted a quantitative, comparative methodology consisting of three stages.

3.1 Problem Selection

A set of 30 problems was curated from the LeetCode platform. The selection was stratified to ensure a balanced sample: 10 "Easy," 10 "Medium," and 10 "Hard" problems. The set was further diversified to cover a range of common computer science topics.

3.2 Data Collection

For each of the 30 problems, a solution was generated using ChatGPT version 4.1, accessed via an embedded agent within the Visual Studio Code editor. A standardized, non-optimized prompt was used for each problem to request a solution in JavaScript. The first complete solution provided by the model was submitted to the LeetCode online judge without any human modification. The resulting runtime (`time_pct`) and memory usage (`space_pct`) percentile rankings were recorded.

3.3 Analysis Protocol

The collected percentile rankings serve as my primary data. A ranking of 80% means the solution was faster or more memory-efficient than 80% of all other successful submissions.

Benchmark: LeetCode provides percentile rankings for both runtime and memory usage by comparing each submission to all other successful submissions for the same problem. Since the vast majority of these are from human users, the percentile metric inherently benchmarks AI-generated solutions against a large, diverse pool of human developers. Thus, a percentile of 80% means the solution is faster (or more memory-efficient) than 80% of all human submissions for that problem. This built-in comparison forms the baseline for this experiment, eliminating the need for a separate human control group.

4 Results

A significant preliminary finding, before analyzing performance percentiles, was the reliability and speed of the solution generation. For all 30 problems, regardless of difficulty, the first solution generated by ChatGPT was accepted by the LeetCode judge on the first submission. This 100% first-try acceptance rate demonstrates a high degree of functional correctness for the given problems.

The data revealed a clear distinction between ChatGPT's runtime performance and its memory efficiency.

4.1 Aggregate Runtime Performance

The AI-generated solutions were exceptionally fast. The aggregate runtime performance across all 30 problems was significantly higher than the median developer.

- The mean runtime percentile was 76.74%.
- The median runtime percentile was 87.04%.

4.2 Aggregate Memory Performance

In contrast, the aggregate memory efficiency of the solutions was average, hovering just above the 50th percentile baseline.

- The mean memory percentile was 54.36%.
- The median memory percentile was 54.32%.

4.3 Performance by Problem Difficulty

Disaggregating the results by problem difficulty uncovered a strong, inverse correlation between problem complexity and runtime performance. While still performing above average, the AI's speed advantage diminished as problems became harder. Memory performance did not show a similar clear trend.

Figure 1 provides a visualization of the raw runtime and memory percentiles for each problem, grouped by difficulty. This scatter plot highlights the spread and clustering of performance across the dataset.

Table 1: Performance Percentiles by Difficulty

Difficulty	**Avg. Runtime**	**Median Runtime**	**Avg. Memory**	**Median Memory**
Easy (n=10)	95.34%	100.00%	48.49%	54.32%
Medium (n=10)	68.80%	78.42%	57.81%	56.13%
Hard (n=10)	61.93%	58.42%	54.76%	52.51%

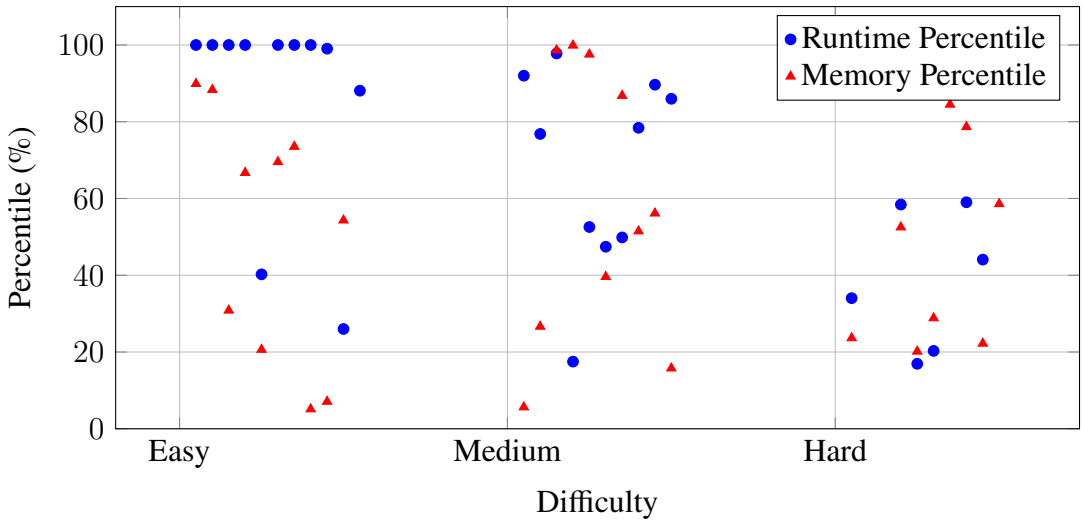

Figure 1: Scatter plot of raw runtime and memory percentiles by difficulty (jittered for visibility)

5 Discussion

5.1 Interpretation of Findings

The results of this study are twofold. First, and most clearly, ChatGPT excels at generating code that is highly optimized for execution speed. A median runtime percentile of 87.04% suggests that its "out-of-the-box" solutions are faster than the vast majority of human submissions on LeetCode. This is likely because the model has been trained on a massive corpus of optimized code, allowing it to recognize and implement efficient algorithms for common problems.

Second, this speed advantage is not absolute and is highly dependent on task complexity. The dramatic drop in average runtime performance from 95.34% on "Easy" problems to 61.93% on "Hard" problems is the most significant finding of this paper. It indicates that while ChatGPT can produce near-perfect solutions for standard, well-defined problems, its ability to optimize for novel or multi-step algorithmic challenges is less reliable.

In contrast, its memory performance is consistently average. This suggests that the model does not prioritize memory optimization to the same degree as runtime, often settling for standard data structures that are functionally correct but not necessarily the most memory-efficient.

It is critical to contextualize the finding of average memory performance. Unlike a fixed algorithm, an LLM's output is malleable. A developer could trivially follow up with a prompt such as, "Refactor this code to be more memory-efficient," and likely receive a superior solution. This underscores a key insight into the evolving role of developers: the key to leveraging LLMs effectively will not just be in generating initial code, but in managing expectations and iteratively refining the output. Future developers will benefit from strong prompt engineering skills and, critically, from implementing robust test automation for load and performance.

5.2 Limitations

This study has three primary limitations. First, its scope is restricted to performance and does not analyze code quality, security, or maintainability. Second, I only tested the first, unoptimized solution from the AI. Third, the LeetCode ranking system is a "black box," and the exact dataset for comparison is unknown.

6 Conclusion and Future Work

This study provided a focused performance benchmark of ChatGPT on competitive programming tasks. My results indicate that **ChatGPT is a remarkably capable tool for generating highly efficient code in terms of speed, though its solutions are average in memory usage.** The model's primary weakness appears to be a degradation in runtime optimization as problem complexity increases.

This confirms the core thesis of Nascimento et al. (2023): AI is a powerful collaborator, not a replacement for human developers. A human developer's role may shift towards guiding the AI, verifying its output, and performing the complex, nuanced optimization on difficult problems where the AI is less effective.

Future work should expand upon this analysis to include the non-functional requirements. Comparing different LLMs and analyzing the impact of prompt engineering on performance would also be valuable next steps.

7 References

Nascimento, N., Alencar, P., & Cowan, D. (2023). *Comparing Software Developers with ChatGPT: An Empirical Investigation*. arXiv preprint arXiv:2305.11837. `https://arxiv.org/pdf/2305.11837`