Analyzing Binary Search Problem Solutions: Comparing Human vs. LLM-Generated Code Using EffiBench

Teja R. Mandadi
Texas A&M University–San Antonio

Visesh Bentula
Texas A&M University–San Antonio

Umapathi Konduri
Texas A&M University–San Antonio

1 Abstract

LLMs are being used to generate code for algorithmic problems but their efficiency vs human written solutions is unexplored. This paper compares the execution time of binary search implementations written by humans vs LLMs (ChatGPT, Google's Gemini, and Anthropic's Claude). We use EffiBench to benchmark 30 binary search problems (easy, medium, hard) in terms of execution time and memory usage. We ensure fair testing with the same problem prompts and runtime environment. Results show Gemini LLM is the fastest and most efficient overall, even beating human solutions in many cases, while human coded solutions are competitive. ChatGPT is balanced but not always the fastest, and Claude is solid but uses a bit more memory. We conclude by discussing implications for AI assisted programming and future work.

2 Introduction

Large Language Models (LLMs) like OpenAI's ChatGPT, Google's Gemini, and Anthropic's Claude are changing the way software is developed by automatically generating code. These models are widely used to tackle coding challenges and algorithmic problems, including classic tasks like binary search. While most existing research on AI-generated code has focused on accuracy and problem-solving capabilities—using benchmarks such as OpenAI's HumanEval and Google's MBPP—there has been far less attention paid to how efficient this code actually is when it runs. Yet in real-world applications, performance—how fast code runs and how much memory it uses—can be just as important as correctness, especially

when dealing with large-scale systems or limited resources. This gap in evaluation is what motivated our study.

Binary search is a great choice for comparing the efficiency of different solutions. It's a well-known algorithm that searches for an element in a sorted list in (log(O(logn) time by cutting the search space in half repeatedly. Because the algorithm is conceptually simple and its optimal form is widely known, it becomes easier to spot inefficiencies in how it's implemented—whether that's unnecessary steps or suboptimal logic. Binary search also shows up frequently in coding interviews and competitive programming, often with variations like searching in rotated arrays, finding bounds, or locating peaks. Despite its relevance, few studies have directly compared human-written and LLM-generated solutions for this problem from an efficiency standpoint across varying difficulty levels. That's the gap we aim to address.

To do this, we used the EffiBench benchmarking framework, which includes 1,000 algorithmic problems from LeetCode and a set of efficient, verified human solutions. EffiBench measures both correctness and performance, tracking metrics like execution time and memory usage. From this dataset, we selected 30 binary search problems and compared how efficiently solutions from different sources—humans and LLMs—performed. Our goal was to see if modern LLMs can not only solve problems correctly, but also write code that runs efficiently, and to explore how this performance varies based on problem difficulty.

The rest of the paper outlines our benchmarking methodology, presents our findings with detailed analysis, and discusses what these results mean for the future of AI-assisted coding. By looking at both correctness and efficiency, our work adds a valuable perspective to how LLMs are evaluated in programming tasks.

3 Methodology

3.1 Benchmarking Overview

We conducted our experiments using the `EffiBench` benchmarking workflow to ensure consistent and fair evaluation of each solution. All tests were executed in a controlled runtime environment to eliminate external variability and ensure reliable comparisons. Each selected problem was evaluated using the same set of inputs across all solution sources to collect uniform performance data.

Our evaluation focused on three primary metrics for each solution: **execution time**, **average memory usage**, and **peak memory usage**. Based on these, we also computed a composite **Efficiency Score** for each solution, defined as the product of **execution time** and **peak memory** (**Time** $\times$ **Max Memory**), capturing overall efficiency in a single value. A lower **Efficiency Score** corresponds to a faster and more memory-efficient solution.

3.2 Problem Selection

We narrowed our study to **30 binary search problems** sourced from the EffiBench dataset. These problems were carefully selected to ensure an even distribution across three difficulty levels: 10 Easy, 10 Medium, and 10 Hard. By focusing exclusively on binary search, we ensured that all solutions were based on the same underlying algorithmic logic, allowing for consistent and meaningful comparisons in terms of performance. This consistency in problem type removes variability that could otherwise arise from comparing different algorithms. Moreover, by incorporating problems across varying levels of difficulty, we were able to explore whether differences in efficiency between human-written and LLM-generated solutions become more noticeable as the problems increase in complexity or involve more edge-case-heavy scenarios.

3.3 Solution Sources

For each problem, we compared solutions from four sources: a human-crafted solution and three LLM-generated solutions. The human solution for each problem was the **canonical, optimized code** provided (or verified) by the **EffiBench** dataset, typically implementing binary search in its most efficient form. The LLM-generated solutions were obtained from **ChatGPT**, **Gemini**, and **Claude** — representing a range of advanced, contemporary LLMs. Each model was prompted with the full problem description (as presented on LeetCode), along with a **consistent prompt template** from the **EffiBench** repository. This prompt format was applied uniformly across all models to avoid bias and included any specific input-output format requirements. The LLMs generated code in response to these prompts, which we then tested for **correctness**.

3.4 Ensuring Correctness

We first verified that each LLM-generated solution produced correct outputs on a set of 10 test cases per problem (the same test cases used to validate human solutions). If a model's solution was incorrect (i.e., it failed one or more test cases), it was either omitted from the efficiency comparisons or corrected if the fix was straightforward. (In our experiments, a few LLM outputs initially failed certain edge-case tests; for example, in one problem involving matrix searches, some LLM solutions did not handle an **empty-array** condition properly. These instances were documented, and only the **correct solutions** were included in the benchmarking phase to ensure fairness in measuring efficiency.) After completing this validation step, we proceeded with performance benchmarking on the **verified correct solutions**.

3.5 Performance Measurement

Each solution was executed in an identical environment (with the same hardware and software configuration) to ensure consistent measurement of runtime and memory usage. **Execution time** was measured in seconds using high-resolution timers, while **memory usage** was tracked in megabytes (MB) using system profiling tools. We recorded both the **average**

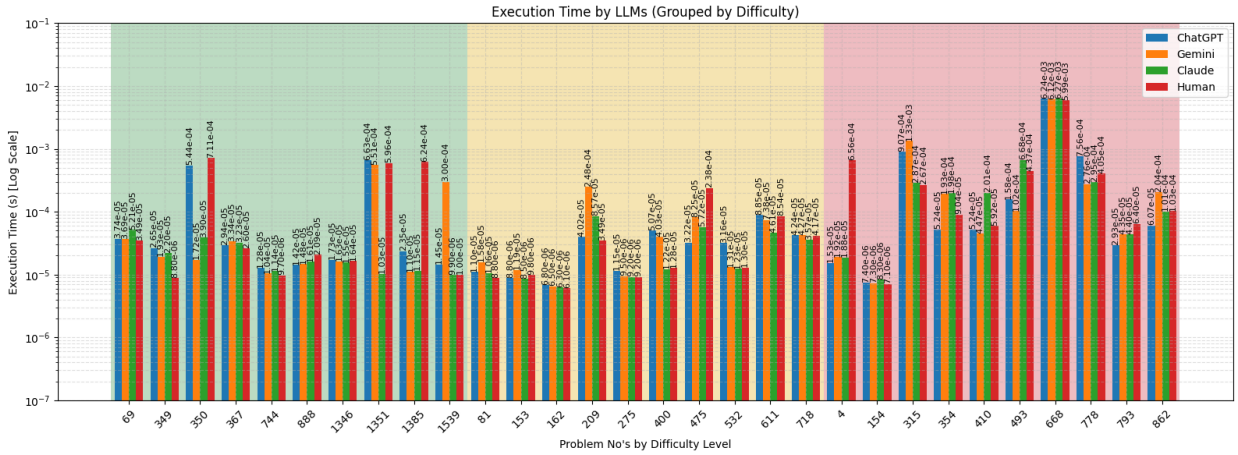

Figure 1: **Execution Time by LLMs (Grouped by Difficulty). Each group of bars corresponds to one problem (x-axis ordered by increasing difficulty from Easy to Hard). Bar height (log scale) indicates execution time in seconds for ChatGPT (blue), Gemini (orange), Claude (green), and Human (red) solutions. Lower is better (faster). Gemini achieves the shortest execution time on most problems, with human and ChatGPT generally close behind, and Claude slightly slower in some cases.**

memory consumed during execution and the **peak memory footprint** reached at any point. To improve accuracy, each solution was executed multiple times, and the **median execution time** was used to reduce the impact of fluctuations caused by transient system load. Memory measurements were taken throughout the execution period to accurately capture the true peak usage. All metrics were systematically logged for subsequent analysis.

3.6 Data Analysis

We aggregated the results for the 30 problems and visualized them to facilitate comparison across solutions. Our analysis considered each metric individually — **execution time**, **memory usage**, and the combined **Efficiency Score**. The results were plotted with problem indices along the x-axis (grouped by **difficulty level** for clarity) and the corresponding metric values along the y-axis. Since execution times varied by several orders of magnitude across different problems and models, we applied a **logarithmic scale** to both the time and efficiency score plots to make variations more visible. To further aid interpretation, the difficulty levels — **easy**, **medium**, and **hard** — were highlighted using background shading in the charts, colored green, yellow, and red respectively. This made it easier to spot trends that correlate with problem complexity.

4 Results

As shown in Figure 1, **Gemini consistently achieves the lowest execution time** for most problems, often outperforming even human solutions. ChatGPT and the human-written

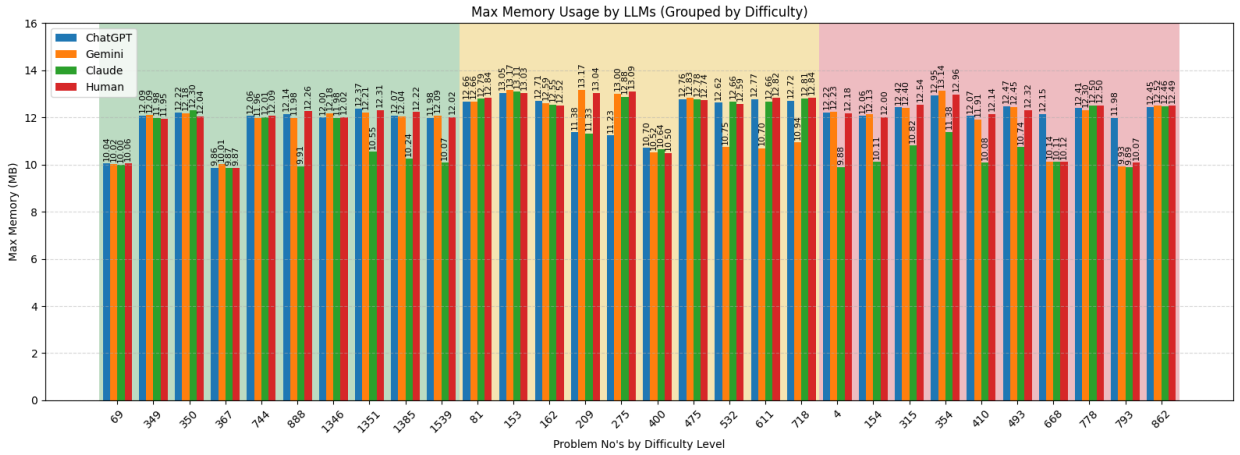

Figure 2: **Peak Memory Usage by LLMs (Grouped by Difficulty). Bar height indicates the peak memory (MB) consumed by each solution on each problem. All methods use similar amounts of memory (around 10–13 MB). Human solutions (red) are slightly lower in some cases, while LLM-generated solutions (blue, orange, green) are marginally higher, with Claude sometimes using the most memory.**

code exhibit roughly comparable runtimes in many cases – their bars are often of similar height – while **Claude's execution times are occasionally higher (slower)** than the others. The differences are relatively small on easy problems, where all methods run in microseconds, but become more noticeable on harder problems. For some hard instances, one can see an order-of-magnitude gap between Gemini (shortest bar) and the slower solutions. Overall, the results highlight Gemini's superior speed, with human and ChatGPT solutions trailing closely behind, and Claude slightly slower in certain cases.

Figure 2 shows that **memory usage across ChatGPT, Gemini, Claude, and human solutions is much more uniform**. Most solutions use roughly 10–13 MB at peak, as indicated by the relatively even bar heights. There is no significant outlier in memory usage on any problem. The human-written solutions sometimes have slightly lower memory footprints (likely because human code may avoid unnecessary data structures), whereas the LLM-generated solutions occasionally show a bit higher memory usage – for instance, Claude (green) has a taller bar than others on a couple of problems, suggesting it may use additional auxiliary structures or less memory-efficient logic in those cases. Overall, however, all four sources are within a close range in terms of memory consumption, and increasing problem difficulty did not lead to large divergences in memory usage.

In Figure 3, which plots the combined efficiency metric, ** Gemini consistently produces the best Efficiency Scores** across the board – its bars are minimal for most problems, reflecting its fast speed coupled with standard memory use. Human solutions and ChatGPT often have comparably low efficiency scores as well, frequently only slightly above Gemini's. Notably, in a few cases (particularly among medium and hard problems), the human solution achieves the best score, indicating an instance where careful human optimization paid off. ChatGPT's scores are usually very close to the human's, showing that it produces relatively

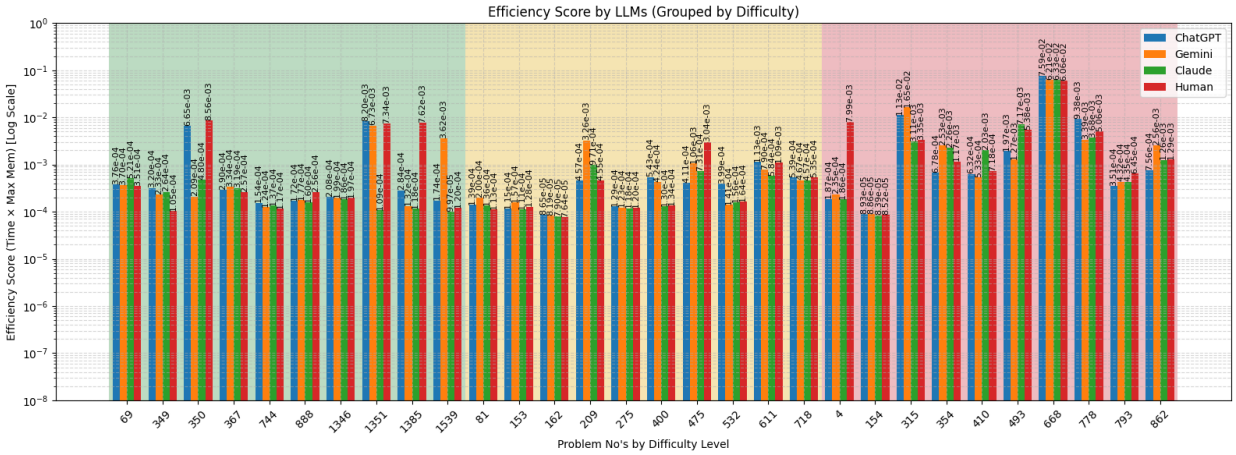

Figure 3: **Efficiency Score by LLMs (Grouped by Difficulty). The Efficiency Score is defined as Execution Time × Peak Memory (lower is better). Bars represent this combined metric for each solution per problem (log scale on y-axis). Gemini's scores (orange) are the lowest in most cases, indicating best overall efficiency. Human (red) and ChatGPT (blue) solutions also achieve low scores close to Gemini's, while Claude (green) tends to have higher scores, denoting slightly worse efficiency.**

efficient code. **Claude's efficiency scores are consistently the highest (worst) among the four** in many problems, aligning with the earlier observations that Claude is a bit slower and sometimes more memory-hungry. Still, all solutions' scores are on the order of 10^{-4} to 10^{-3} for most tasks, so they are all reasonably efficient. The gaps widen on a few specific hard problems where one solution's inefficiency spikes the score (seen as isolated taller bars). In summary, the Efficiency Score comparison emphasizes Gemini's overall lead in efficiency, while also showing that ChatGPT and well-written human code are not far behind, and that Claude, despite being slightly less efficient, remains within the same order-of-magnitude range.

5 Discussion

The benchmarking results provide several insights into the comparative performance of human versus LLM-generated code for the binary search problems:

LLM vs. Human Performance: Our experiments demonstrate that state-of-the-art LLMs can **match or even exceed human programmers in code efficiency** for a well-defined algorithm like binary search. In particular, Google's Gemini model stood out by producing solutions that run extremely quickly – in fact, Gemini's solutions were the fastest in the majority of cases, sometimes by a noticeable margin. This suggests that Gemini's training or inference process may favor more optimized code patterns (perhaps it has learned efficient idioms for binary search). Human-written solutions, which we expected to be near optimal, indeed performed strongly and even outperformed the AI models on a few problems. This typically occurred in more complex scenarios where a human developer might apply a clever

optimization or handle an edge case more efficiently than the generic approaches the LLMs learned. OpenAI's ChatGPT was generally very competitive with the human solutions on both time and memory metrics, indicating that it usually produces quite polished code for binary search. Anthropic's Claude, while still performing well, was slightly less efficient – for example, its code sometimes had extra steps or used more memory (as reflected in the marginally higher memory usage and efficiency scores). These differences, however, were not extreme; all approaches achieved fast runtimes and modest memory usage suitable for the problem sizes.

Impact of Problem Difficulty: Problem difficulty influenced the spread of performance outcomes. For **easier problems**, all solutions — both human-written and LLM-generated — executed almost instantaneously and consumed minimal memory, leaving little opportunity for one approach to significantly outperform the others. As the problems increased in difficulty (i.e., **medium** and **hard**), often involving more complex input scenarios or edge cases, we observed greater variability in results. In some hard problems, inefficiencies became apparent — for instance, an LLM might not handle a special case as efficiently, resulting in extra loop iterations or additional memory overhead. In one such hard problem (identifying the k weakest rows in a matrix), the human solution applied an **optimal strategy** that all LLMs initially missed. As a result, the LLM-generated solutions either failed certain test cases or took slightly longer to execute. This highlights that **LLMs, despite their capabilities, can still struggle with niche or edge-case scenarios**, and that problem difficulty can amplify the impact of suboptimal logic. Nonetheless, for most hard problems in our set, at least one LLM — often **Gemini** — produced a solution with efficiency comparable to the human-written one, showcasing the ongoing advancement in AI-generated code.

Code Characteristics and Efficiency: Upon manual inspection of several solutions, we identified patterns that help explain the observed differences in efficiency. The **human-written solutions** were typically succinct and purpose-built — often implementing binary search directly, with minimal extra code. In contrast, the **LLM-generated solutions** occasionally included superfluous operations (e.g., converting data types or invoking library functions where a more direct approach would suffice), which introduced slight overhead.

From a memory standpoint, all solutions had broadly similar requirements, as the tasks themselves were not inherently memory-intensive. However, when an LLM produced a more **convoluted implementation** — such as creating unnecessary data copies or relying on non-optimal data structures — it tended to consume more memory. The slightly higher memory usage observed in some of **Claude's** solutions may be attributed to these patterns.

It is also important to note that none of the LLMs were explicitly instructed to optimize for performance; they were prompted solely to solve the problem correctly. The fact that their performance came close to optimal suggests that modern LLMs often generate reasonably efficient code **by default** for well-defined algorithms. Still, the small performance gaps we observed indicate that there is room to further **fine-tune or guide LLMs for optimization**.

Implications: These findings have meaningful implications for both users and developers of code-generation AI. For practitioners integrating **LLMs** into their development workflows, our results are encouraging: an LLM such as **ChatGPT** or **Gemini** can not only produce correct solutions but also deliver efficiency that, in many cases, is comparable to that of a

human expert. However, developers should remain vigilant when handling edge cases and should evaluate the performance of **AI-generated code**, particularly on more complex or non-trivial problems.

From a research perspective, our study reinforces the importance of benchmarks like **EffiBench** that extend beyond correctness and include performance-based evaluation. As LLMs continue to evolve, **efficiency could become a key differentiator** — users may increasingly favor models that not only solve problems accurately but also generate code that executes faster and consumes less memory. This trend could drive the development of new training objectives or **fine-tuning strategies** that explicitly incorporate runtime performance as part of model optimization.

Furthermore, our results suggest the potential for **hybrid approaches**, where an LLM's initial output is postprocessed or refined using external tools to combine the strengths of both AI-generated correctness and human-level performance tuning — achieving the best of both worlds.

6 Conclusion

We presented a comparative study of **human-written** and **LLM-generated** solutions for binary search problems, using a rigorous benchmarking methodology. Our evaluation across 30 problems shows that **LLM-generated code can achieve performance on par with human-written code**, and in the case of Google's *Gemini*, often surpasses it in terms of execution speed and overall efficiency. Human solutions continued to perform strongly — in some cases even outperforming all LLMs — highlighting the value of human insight and manual optimization. OpenAI's *ChatGPT* offered a balanced trade-off between speed and memory usage, making it a reliable overall performer, though not the absolute fastest. Anthropic's *Claude* produced correct and generally solid results but tended to trail slightly behind the others in efficiency, particularly in terms of memory usage.

These findings emphasize that as **LLMs become more integrated into software development**, evaluation criteria must evolve to include **efficiency alongside correctness**. Our use of the **EffiBench** framework demonstrates a structured way to assess both dimensions. Looking ahead, we plan to broaden our analysis to include a wider variety of algorithms beyond binary search — such as sorting, graph traversal, and dynamic programming — and to incorporate more recent models like *GPT-4* and other emerging systems. We also aim to explore techniques for improving LLM efficiency, such as optimization-aware prompts or hybrid approaches that combine LLM-generated outputs with **static analysis tools** to identify and correct inefficiencies.

Ultimately, we envision a future where **human–AI collaboration** yields solutions that are not only correct and maintainable, but also optimized for performance — merging the best of both human expertise and AI automation.