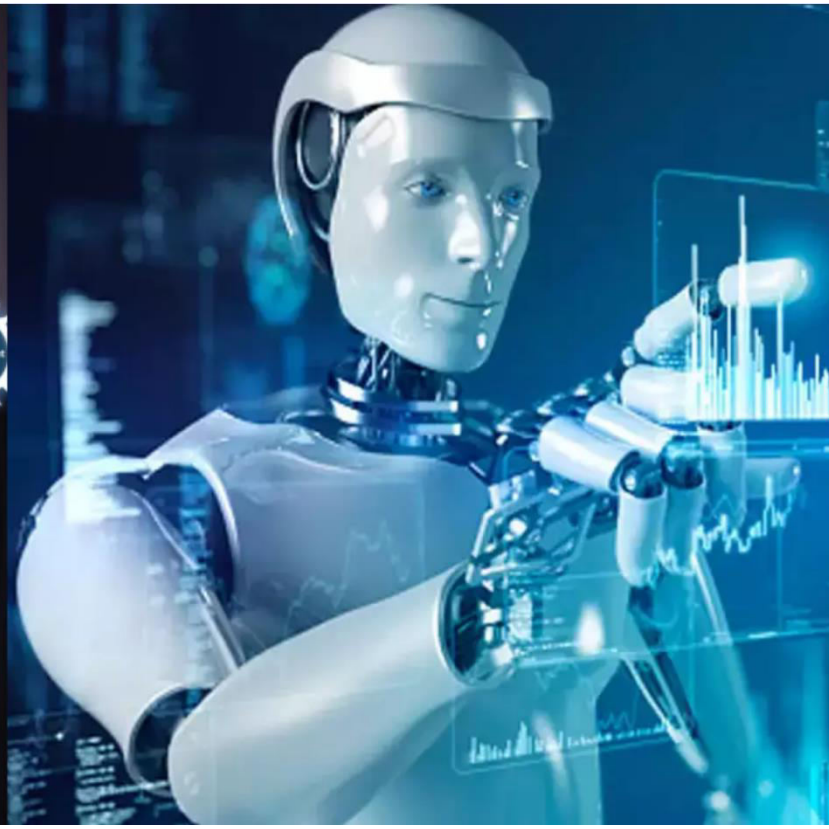
IJIRSET
International Journal of Innovative Research in Science | Engineering | Technology

ISSN(O): 2319-8753
ISSN(P): 2347-6710

International Journal of Innovative Research in Science Engineering and Technology (IJIRSET)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 8.699

Volume 15, Issue 3, March 2026

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503301|

Empirical Evaluation of Runtime Efficiency and Memory Wall: A Comparative Study of Python 3.13.0 and C in Resource Constrained Environments

Achintya Gandra

Independent Researcher and Product Developer, Warangal, Telangana, India

ABSTRACT: The continuous evolution of software development heavily favours high-level interpreted languages like Python for rapid prototyping, often at the expense of raw computational efficiency. This paper presents an empirical comparative study evaluating the execution latency and peak memory consumption of Python 3.13.0 against the compiled C language within a resource-constrained environment (8GB RAM workstation). Utilizing the Sieve of Eratosthenes algorithm across dataset sizes ranging from 100,000 to 30,000,000 elements, this study quantifies hardware limitation thresholds, specifically focusing on the "Memory Wall." Experimental results demonstrate that while C maintains a linear resource trajectory (peaking at approximately 35 MB), Python incurs a significant memory overhead, hitting a critical performance bottleneck at 10 million elements where execution time and RAM consumption scale exponentially. Ultimately, this research validates that for memory-intensive operations on standard consumer hardware, low-level languages like C remain indispensable despite their higher development overhead.

KEYWORDS: Performance Benchmarking, Comparative Study, Memory Wall, Execution Latency, Python 3.13, C Programming, Resource-Constrained Environments**.**

I. INTRODUCTION

Modern software development is continuously evolving towards higher-level and more abstract programming approach. This shift in pattern from low-level languages, such as C, to high-level languages, such as Python, has prioritized developer productivity over raw machine efficiency. In current ecosystem, Python 3.13.0 has emerged as a dominant language for rapid prototyping, data science, and automation, significantly reducing the **"Time-to-Market"** for independent developers.

However, this gain in productivity introduces a critical trade-off. While high-level languages offer a reduction in coding complexity, they often encounter a **"Memory Wall"** and increased execution latency when deployed on resource-limited hardware. Conversely, low-level languages like C, despite their higher development overhead and the requirement of manual memory management, maintain superior execution efficiency and a minimal hardware footprint.

A significant gap exists in current literature, as most existing performance benchmarks focus on high-end server environments or specialized clusters. There is a notable lack of primary empirical research on how these runtimes behave on mid-range consumer hardware, such as 8GB RAM laptop, which represents the standard workstation for many students and independent developers worldwide.

This study provides a quantitative investigation into why the industry continues to prefer low-level language like C for performance-critical systems despite the rapid development capabilities of Python 3.13.0. Through controlled experiments on a standardized 8GB RAM testbed, this study measures the computational overhead of higher-level programming language.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503301|

The remainder of this Paper is organized as follows: Section II review existing literature on language performance; Section III details the experimental methodology and hardware specifications; Section IV presents quantitative results and comparative visualizations and Finally, Section V concludes the study with a discussion on industrial implications.

II. LITERATURE SURVEY

The architectural distinctions between compiled and interpreted languages are foundational to software engineering and are strictly defined by their governing standards. The ISO/IEC standard **[2]** emphasizes low-level hardware access and manual memory management, which provides granular control over system resources. In contrast, the official Python documentation **[1]** outlines a design philosophy focused on high-level abstraction, dynamic typing, and automated garbage collection to maximize developer productivity.

The disparity in development overhead is further influenced by modern Integrated Development Environments (IDEs). Tools such as Visual Studio Code for Python **[3]** and Code::Blocks for C **[4]** offer tailored environments, yet the inherent complexity of C requires a deeper understanding of compiler toolchains, such as the GNU Compiler Collection (GCC) **[5]**, to achieve optimal machine-level execution.

Existing comparative literature and established benchmarking platforms, such as the Computer Language Benchmarks Game **[6]**, provide extensive empirical data confirming the execution speed advantage of C over Python. However, these standard benchmarks are predominantly executed on high-performance servers or virtualized cloud instances with abundant memory. While Rysak **[7]** analysed logical algorithms such as Tower of Hanoi and Huffman encoding, their study primarily focused on execution speed in isolated logic blocks. Consequently, a significant gap remains in evaluating the 'Memory Wall'—specifically how the resource overhead of interpreted languages scales on standard consumer-grade hardware equipped with restricted memory (e.g., 8GB RAM). This paper extends that research by evaluating memory-intensive operations and large-scale data processing on standard 8GB RAM hardware, identifying the specific **'break points'** where Python and C encounter hardware limitations.

III. METHODOLOGY

A. Experimental Environment

The performance benchmarks were conducted on a standardized workstation representing a typical independent developer's hardware. The system specifications are as follows:

- Processor: 12th Gen Intel(R) Core (TM) i5-1235U (1.30 GHz base, with Turbo Boost).
- Memory: 8.00 GB (7.69 GB usable) DDR4 RAM.
- Architecture: 64-bit Operating System, x64-based processor.

The software stack included the MinGW-w64 GCC compiler for the C implementation and Python 3.13.0 for the interpreted tests. To ensure data integrity, each test case (N = 100K, 1 M, 10M, 20M, and 30M) was executed three times. The arithmetic mean of these trials was recorded to mitigate the impact of background OS processes and thermal throttling.

B. Algorithm: Sieve of Eratosthenes

The Sieve of Eratosthenes was selected as the primary benchmarking algorithm due to its heavy reliance on both CPU cycles and sequential memory access.

1. Initialization: A contiguous data structure (a char array in C and a list in Python) is initialized to size N, where N represents the upper bound of the prime search range (from 100K to 30M).
2. Iterative Elimination: The algorithm implements the standard prime-finding logic: starting from p=2, all multiples p^2, p^2+p, p^2+2p…. are marked as non-prime. This process repeats for all p less than or equal to sqrt(N).
3. Resource Tracking: Execution latency is measured using the system clock, while peak memory usage is monitored via internal profiling tools.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503301|

C. EXECUTION TIME

To ensure high precision and eliminate human error, execution time was calculated programmatically within the code itself.

- Built-in timing libraries (such as <time.h>) were utilized to record a precise timestamp immediately before the main algorithm started executing (Start Time).
- A second timestamp was recorded the exact moment the data processing finished (End Time).
- The overall execution speed was automatically computed by the script using the standard formula: **Total Time Taken = End Time - Start Time**.
- Automating this measurement directly within the code ensures consistent, exact results and completely prevents manual timing discrepancies.

D. MEMORY CONSUMPTION

To ensure a completely fair comparison between C and Python, memory usage was calculated manually for both languages.

- Rather than relying on automated profiling tools—which handle background processes and garbage collection differently depending on the language—the exact memory size of the data structures used in both implementations was calculated manually.
- This approach guarantees that both languages were evaluated on a level playing field, focusing strictly on the actual space required to hold the data structures.

IV. EXPERIMENTAL RESULTS

A. Experiment Data

The benchmark results are organized into the following three tables to provide a comprehensive view of performance:

- **Table 1:** Presents the raw execution times (latency) recorded across three independent trials for both C and Python 3.13.0.
- **Table 2:** Details the raw peak memory consumption data recorded across three independent trials for both implementations.
- **Table 3:** Provides a summary of the arithmetic mean (average) for both peak memory usage and execution time, highlighting the performance gap between the two languages.

TABLE 1: RAW EXECUTION TIME DATA(SECONDS)

Dataset Size (N)	Language	Trial 1	Trial 2	Trial 3
100K	C	< 0.001	< 0.001	< 0.001
	Python	0.024322	0.008318	0.008625
1M	C	0.015	0.015	0.015
	Python	0.110844	0.072010	0.07685
10M	C	0.065	0.063	0.058
	Python	**0.994957**	**0.987445**	**0.989307**
20M	C	0.297	0.181	0.19
	Python	**1.988643**	**2.126295**	**2.132333**
30M	C	0.362	0.342	0.364
	Python	**3.183044**	**3.144842**	**2.901337**

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503301|

TABLE 2: RAW PEAK MEMORY CONSUMPTION DATA (MB)

Dataset Size (N)	Language	Trial 1	Trial 2	Trial 3
100K	C	0.1320	0.1320	0.1320
	Python	0.8442	0.8442	0.8422
1M	C	1.2531	1.2531	1.2531
	Python	8.233	8.233	8.233
10M	C	12.0719	12.0719	12.0719
	Python	**81.9528**	**81.9528**	**81.9528**
20M	C	23.9205	23.9205	23.9205
	Python	**162.7856**	**162.7856**	**162.7856**
30M	C	35.6974	35.6974	35.6974
	Python	**243.4017**	**243.4017**	**243.4017**

TABLE 3: AVERAGE OF PEAK MEMORY CONSUMPTION DATA (MB) AND EXECUTION TIME DATA (SECONDS)

Dataset Size (N)	Language	Average Time (seconds)	Average Memory
100K	C	< 0.01	0.1320
	Python	0.013	0.8442
1M	C	0.015	1.2531
	Python	0.086568	8.233
10M	C	0.062	12.0719
	Python	**0.99**	**81.9528**
20M	C	0.22	23.9205
	Python	**2.082**	**162.7856**
30M	C	0.356	35.6974
	Python	**3.07**	**243.4017**

Fig 1 & Fig 2: illustrates the logarithmic graph of exponential growth in Python's memory consumption as the dataset size (N) increases, compared to the near-constant performance of C.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503301|

FIG 1

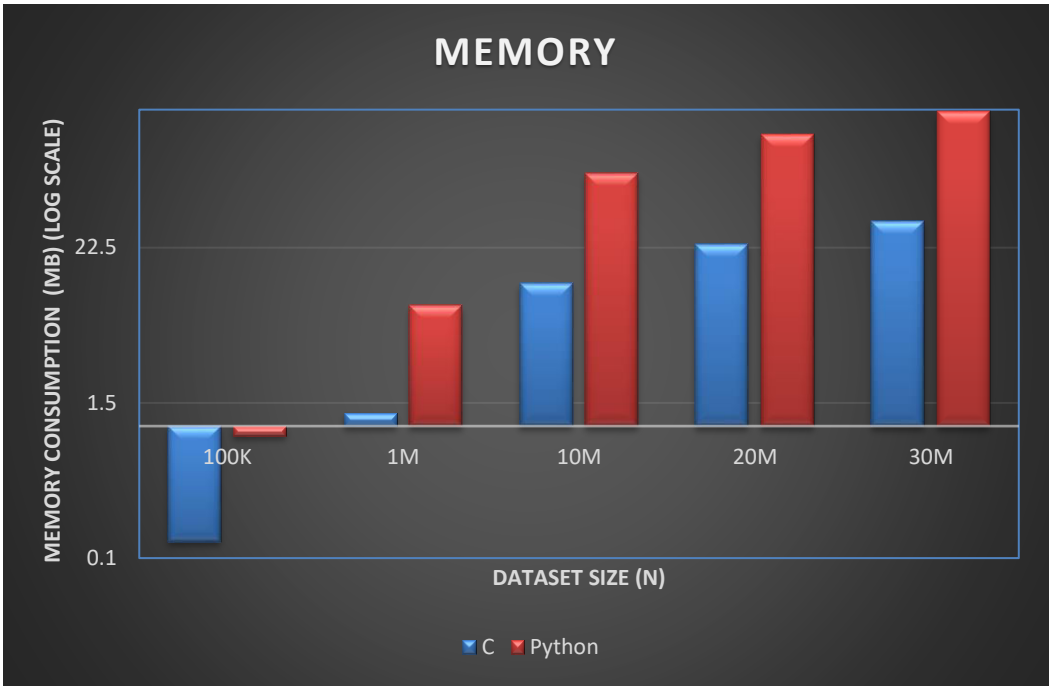


FIG 2

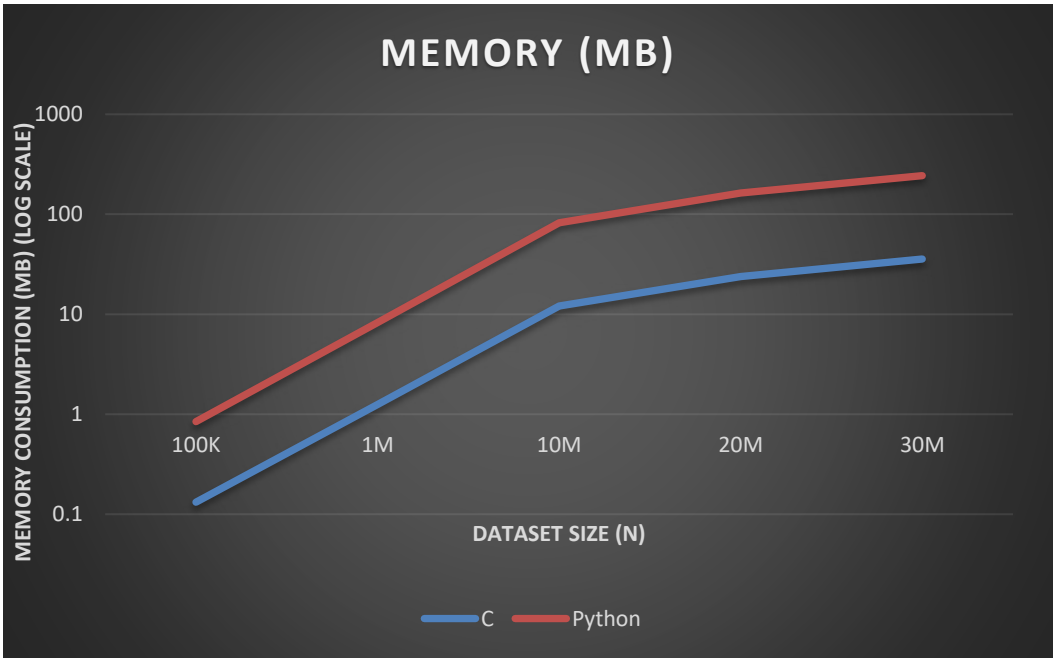


Fig 3 & Fig 4: illustrates the logarithmic graph of exponential growth in Python's execution time as the dataset size (N) increases, compared to the near-constant performance of C.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503301|

FIG 3

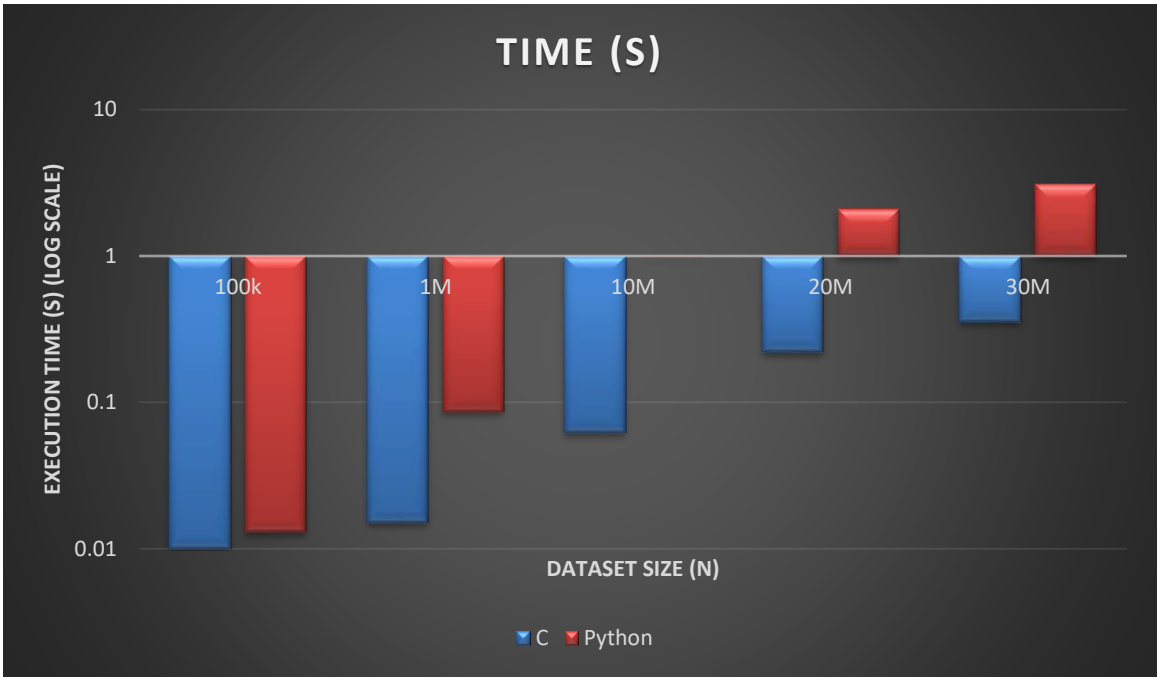


FIG 4

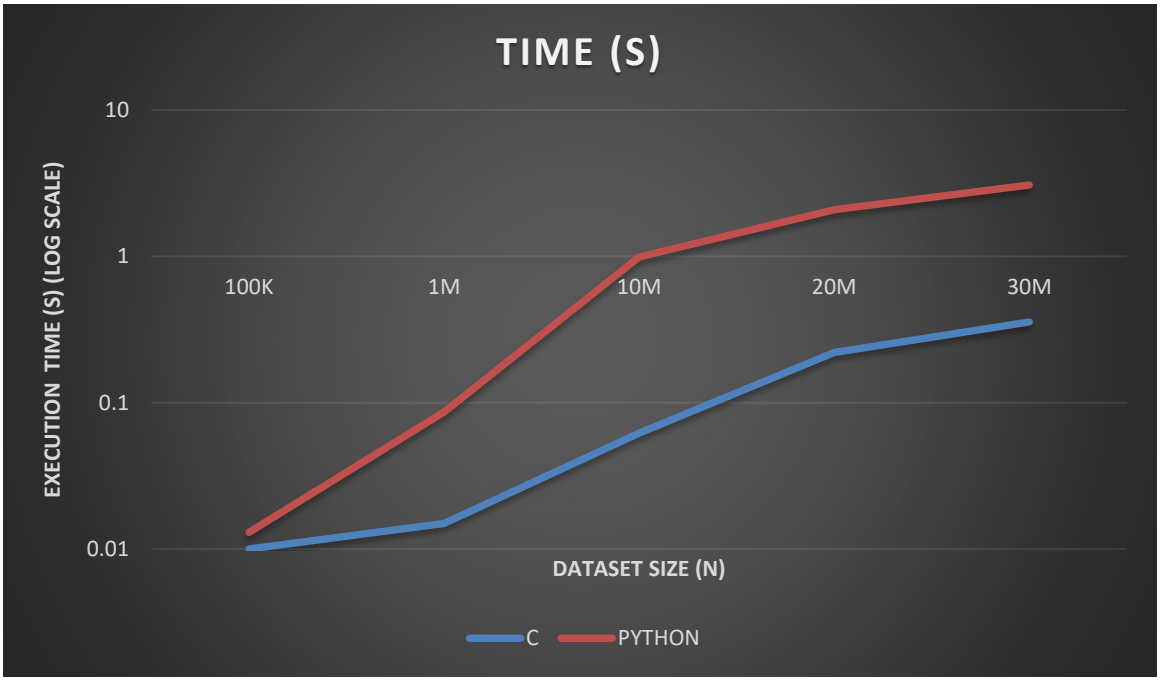


Fig 5: provides a side-by-side logarithmic graph of C and Python execution time and memory consumption data.

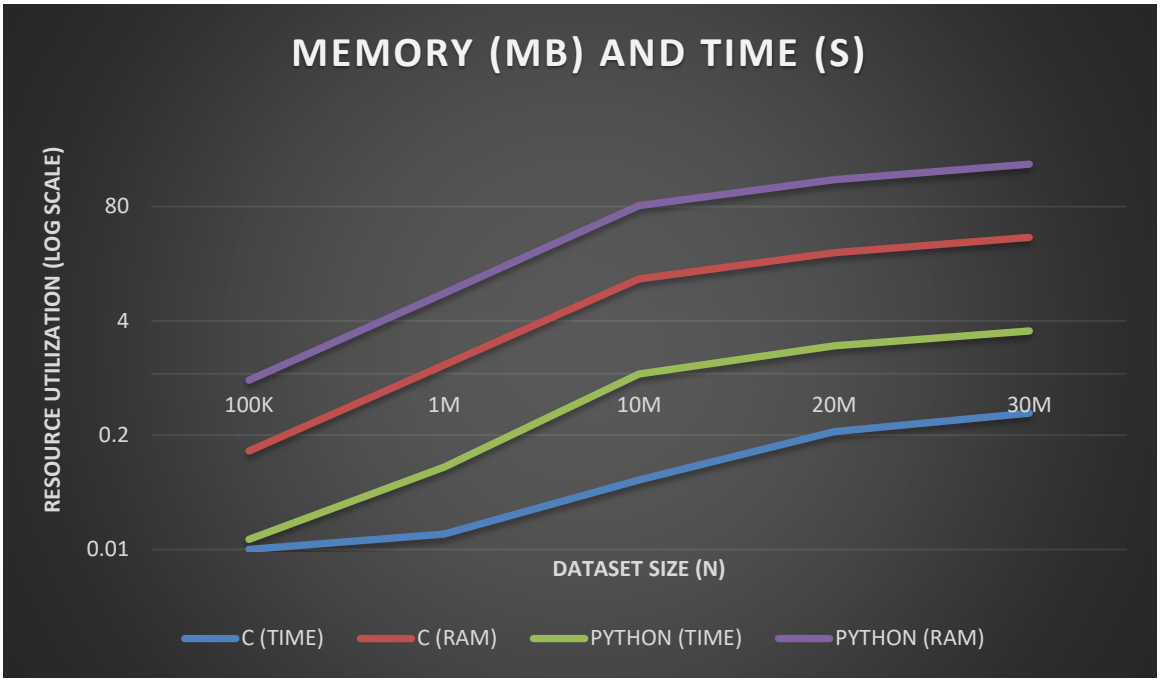
International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503301|



V. CONCLUSION

A. Summary of Findings:

This study conducted an empirical comparison between C and Python 3.13.0, focusing on the "Memory Wall" encountered during large-scale prime number generation. The results consistently demonstrate that while Python offers superior developer productivity, it incurs a significant performance penalty on mid-range hardware (8GB RAM).

Key observations include:

- **Baseline Overhead:** Python exhibits a constant execution latency even for small datasets (100K), whereas C remains below the measurable threshold.
- **Memory Scaling:** At N=30M, Python's memory footprint is approximately 8x to 10x larger than C's, nearing 250 MB of consumption for a task that C completes using only ~35 MB.
- **The Break Point:** The "Memory Wall" becomes visible at 10M elements, where Python's execution time spikes exponentially due to object management and garbage collection overhead.

B. Final Verdict:

For independent developers and researchers working within the constraints of 8GB RAM workstations, C remains the "Master" language for memory-intensive, low-level computational tasks. Python 3.13.0 is ideal for rapid prototyping, but its "Time-to-Market" advantage is offset by heavy resource demands when scaling beyond 10M data points.

C. Future Work:

1. Algorithmic Diversity and Scaling:

This experiment focused exclusively on the **Sieve of Eratosthenes** due to its specific memory access patterns. Future research could expand this to include non-linear algorithms or I/O bound tasks to see if the Python performance gap remains consistent. Additionally, while the 30M dataset size exposed a clear bottleneck, future tests could push the boundaries toward the full 8GB system limit to observe complete system failure points.

2. Software Advancements (Python 3.13):

A critical area for future work is the evaluation of **Python 3.13's "No-GIL" (Free-Threading)** builds. As the industry moves toward multi-core dominance, testing how a GIL-free environment handles large data structures could reveal if Python can close the performance gap with C in parallel processing tasks.

International Journal of Innovative Research of Science, Engineering and Technology (IJIRSET)

Impact Factor 8.699

|www.ijirset.com |A Monthly, Peer Reviewed & Refereed Journal| e-ISSN: 2319-8753| p-ISSN: 2347-6710|

Volume 15, Issue 3, March 2026

|DOI: 10.15680/IJIRSET.2026.1503301|

3. Optimized Python Libraries:

This paper evaluated "Pure Python" to measure the baseline overhead of the interpreter. Future studies should contrast these results against specialized C-extension libraries like NumPy or Pandas to determine if a "Pythonic" environment can achieve C-level performance through external binary optimization.

REFERENCES

[1] Python Software Foundation, "Python 3.13 Documentation," 2024. [Online]. Available: https://docs.python.org/3.13/.

[2] ISO/IEC, "ISO/IEC 9899:2024 Information Technology-Programming languages-C". [Online]. Available: https://iso.org/standard/82075.html

[3] Microsoft, "Visual Studio Code User Guide: Python Development," 2026. [Online]. Available: https://code.visualstudio.com/docs/languages/python

[4] Code::Blocks Team, "Code::Blocks User Manual and Feature Documentation," 2026. [Online]. Available: https://www.codeblocks.org/user-manual/

[5] Free Software Foundation (FSF), "GCC, the GNU Compiler Collection - Documentation," 2026. [Online]. Available: https://gcc.gnu.org/onlinedocs/

[6] I. Gouy, "The Computer Language Benchmarks Game: Python 3 vs C gcc," 2026. [Online]. Available: https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html/

[7] Rysak, P. (2023). Comparative analysis of code execution time by C and Python based on selected algorithms. Journal of Computer Sciences Institute, vol.*26*, pp. 93–99, 2023. [Online]. Available: https://doi.org/10.35784/jcsi.3109

INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi cross ref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN SCIENCE | ENGINEERING | TECHNOLOGY

9940 572 462 6381 907 438 ijirset@gmail.com

www.ijirset.com

Scan to save the contact details