Journal of Industrial Information Technology and Application *ISSN(Online): 2586-0852*
Vol.9 No. 1

The Price of Convenience: Empirical Runtime Study on Type Casting Across Programming Paradigms

Ashton Curry [1,], Rane Murphy [1], Ka Lok Man [2], Yuxuan Zhao [3], and Kamran Siddique [1]*

[1]Dept. of Computer Science and Engineering, University of Alaska Anchorage
[2]Department of Computing, School of Advanced Technology, Xi'an Jiaotong-Liverpool University
[3]School of AI and Advanced Computing, Xi'an Jiaotong-Liverpool University

Abstract. This study examines the runtime impact of implicit vs. explicit type casting across five programming languages (Python, JavaScript, C#, C++, and Java). Experiments reveal significant differences, with compiled languages generally performing faster, highlighting trade-offs between developer convenience and runtime efficiency.

Keywords; Programming languages; Runtime Efficiency; Paradigms

Cite this paper as : Ashton Curry, Rane Murphy, Ka Lok Man, Yuxuan Zhao, and Kamran Siddique (2025) "The Price of Convenience: Empirical Runtime Study on Type Casting Across Programming Paradigms", Journal of Industrial Information Technology and Application, Vol. 9. No. 1, pp. 1032 - 1038

*Corresponding author : ksiddique@alaska.edu
Received: Nov. 11. 2024 Accepted: Jan. 8. 2025 Published: Mar. 31. 2025

1. Introduction

Modern programming languages increasingly aim to resemble natural languages, with many adopting dynamics typing to let developers declare variables without specifying data types. While this convenience shortens development time, it raises questions about its impact on program performance. The ongoing debate in the computer science community [2] explores static and dynamic type systems, comparing their performance at runtime and during program creation. Static typing performs type checking at compile time, while dynamic typing defers it to runtime, affecting debugging and error detection [3]. Although static and dynamic-typed languages are compared at runtime in [1], the impact of conveniences like implicit type casting remains less explored. As a subset of dynamic typing, implicit declaration allows the language to determine and adjust data types based on context throughout the program. This paper presents an experimental analysis of implicit and explicit type casting across five programming languages: interpretive (Python, JavaScript), compiled (C#, C++), and a middle-man language. It tests the hypothesis that explicit variable declaration, though less programmer-friendly, leads to faster runtimes, with compiled languages performing the best.

2. Research Methods

A. General Description

The experimental setup involved 8 different small programs. Their purposes will be summed up in this list:

- Explicit Declaration: declare a variable to its specific type (e.g. ‘int x’).
- Implicit Declaration: declare a variable implicitly (e.g. ‘x = 1’).
- Explicit Float Addition: add two explicitly declared float variables together.
- Mixed Float Addition: add an implicit and explicit variable together.
- Explicit Integer Addition: add two explicitly declared integer variables together.
- Mixed Integer Addition: add an implicit and explicit integer together.
- Addition of explicit like-types then mixed types: two floats are added together, followed by two integers, then an integer and a float are added together

- Addition of explicit mixed types then like-types: a float and an integer are added together, followed by another float and integer added together. Then, the two results are added together.

Each experiment involved running a code loop for 1,000 iterations to gather sufficient data, with the mean calculated for cross-language and experiment comparisons. Mixed and implicit additions/declarations represent runtimes for implicit type casting, while explicit additions/declarations represent explicit type casting. The final two experiments measured whether runtime is faster for explicit addition (same variable types) or implicit addition (different variable types) using explicit type declarations.

B. Implementation Details

C#, C++, Java, Python, and JavaScript accomplished running each experiment using their own ways of implicit and explicit declaration. Python and JavaScript default to implicit declaration so a different approach was used to simulate both languages' explicit declaration. The implementation details for each programming language will be described.

C# supports both implicit and explicit variable type declarations but imposes restrictions, such as requiring separate statements for implicitly typed variables and explicit casts for adding floating-point values to integers. Runtime was measured with the Stopwatch class, and data was exported to Excel. C++ offers similar type declaration support but with fewer restrictions, allowing more flexibility. Timing was captured with the clock function and results exported similarly.

Java balances features of C# and C++. Like C#, it disallows compound implicit declarations but permits adding floats to integers with a warning. Runtime was recorded using nanoTime and exported to Excel. Python required workarounds for explicit type declarations using float() and int(), with timing handled by timeit and data stored in Excel using pandas. Similarly, JavaScript uses libraries like performanceObserver for timing and ExcelJS for data export, with explicit typing achieved through Number() and parseInt().

3. Experimental Results or Performance Evaluation

Figure 1. shows the mean runtime comparison between C and C#. For all C experiments, each runtime is falling within the range of 2.21 ns to 2.96 ns. The mean runtimes for C++ are from 1.93 ns to 2.77 ns. The experiment categories of explicit vs

implicit declaration and explicit vs mixed integer arithmetic showed negligible difference in mean runtime. Explicit vs mixed float arithmetic and likewise then mixed vs mixed then likewise type addition showed a notable difference in mean runtime. Both languages ran similarly in the nanosecond magnitude range. C++ has a mean runtime lower than that of C in every experimental category. Java exhibited distinct runtime behavior during the experiment, with spikes and subsequent drops to lower values over two periods in 1,000 iterations, necessitating a different analysis approach. Stable windows (trials 1–55 and 75–175) were identified for analysis, with values becoming negligible after trial 235. Figure 2 shows that early Java trials generally contribute to higher mean runtimes, except for the implicit declaration experiment and mixed type addition experiment, where trials 75–175 had lower or similar mean runtimes. Figure 3 illustrates Python's and Javascript mean runtimes. Python's range is from 0.59 µs to 1.05 µs. Differences in mean runtime were negligible for some categories (e.g., mixed float arithmetic), but explicit vs. implicit declaration and mixed integer arithmetic showed notable variations. JavaScript's mean runtimes vary from 151.80 µs to 1716.23 µs, with notable differences in some categories, such as explicit vs. implicit declaration. Therefore, Python consistently outperformed JavaScript, operating in the microsecond range, except for JavaScript's explicit declaration, which shifted to the millisecond range.

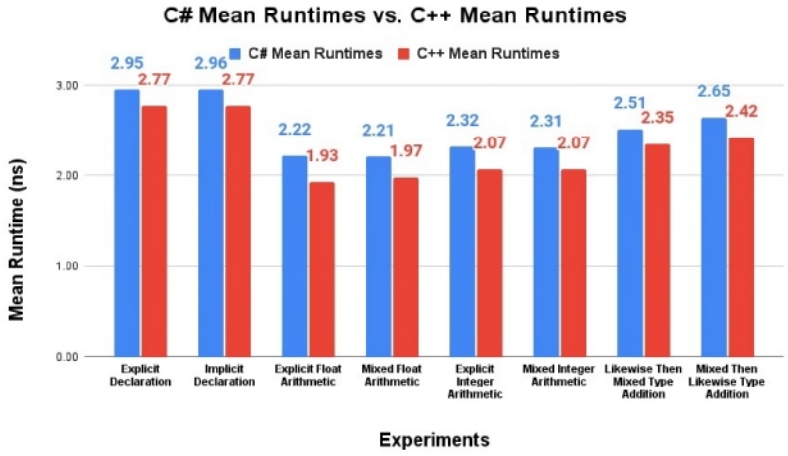

Figure 1. C# vs. C++ Mean Runtimes

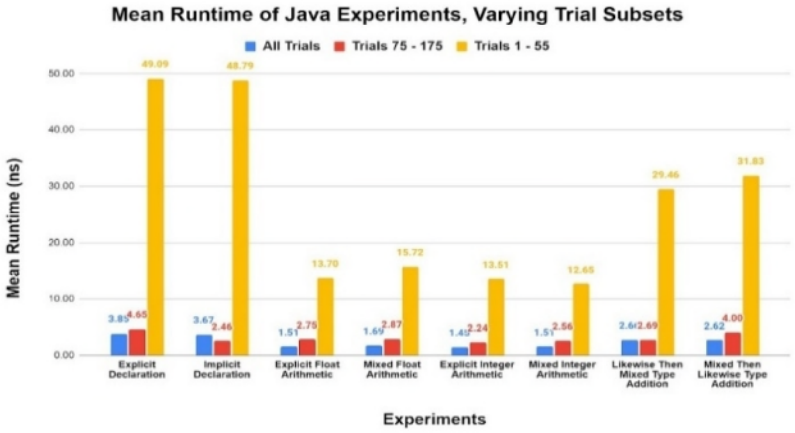

Figure 2. Mean Runtime of Java Experiments, Varying Trial Subsets

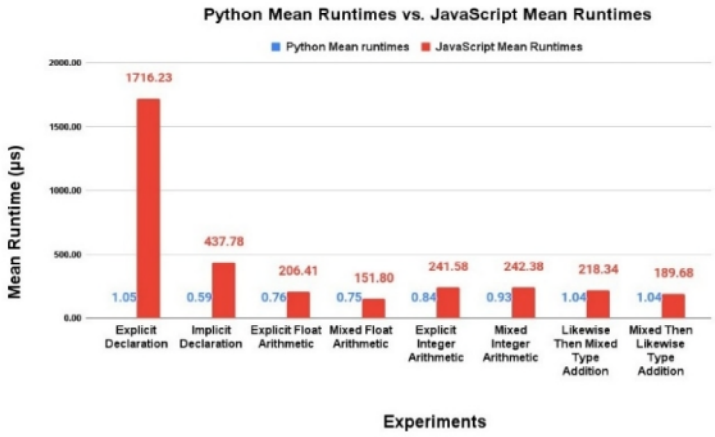

Figure 3. Python vs. JavaScript Mean Runtimes

Figure 4. shows all languages compared against one another by experiment. At a glance, we can confirm part of our hypothesis that compiled languages would have faster runtimes. The compiled languages enjoy a mean runtime several orders of magnitude faster than the closest interpreted language, Python. Furthermore, Python had a mean runtime several orders of magnitude faster even than JavaScript. Additionally, we can see that the compiled languages all have a very close mean runtime within one order of magnitude with one another.

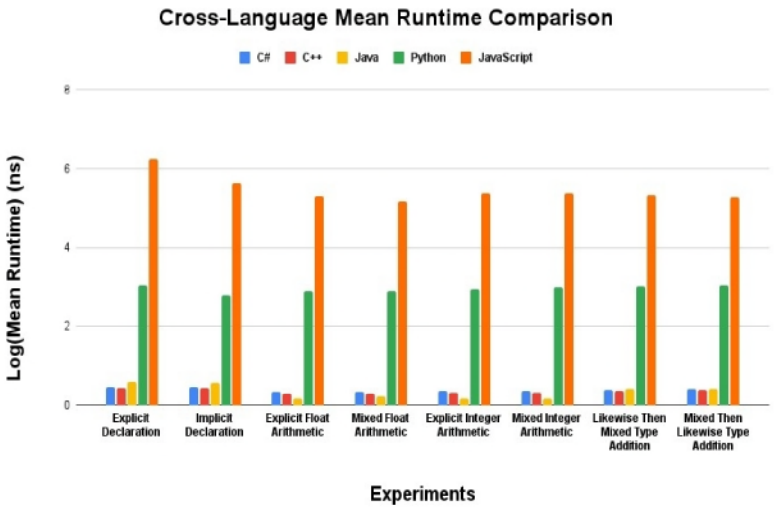

Figure 4. Mean Runtime of Java Experiments

4. Conclusion

This study investigated runtimes of implicit and explicit type casting experiments across various programming languages and paradigms. The findings reveal substantial differences in runtime performance, with compiled languages like C# and C++ outperforming interpretive languages such as Python and JavaScript. Java, as a hybrid language, demonstrated unexpected and inconclusive results. These results confirm the hypothesis that explicit type declarations, though less convenient during development, generally enhance runtime efficiency, highlighting the trade-offs between coding simplicity and execution performance. For performance-critical tasks, explicit casting in statically-typed languages is recommended, while dynamic typing suits projects prioritizing flexibility. Future research could explore newer languages, updated versions, cognitive impacts, and real-world applications to validate these findings and adapt to evolving technologies.

In conclusion, while this study has shed light on important aspects of type casting in programming, the dynamic nature of programming languages and evolving compiler technologies calls for future research. Developers and language designers should remain adaptable, continually balancing convenience and performance based on the latest empirical evidence.

References

[1] S. Hanenberg, “An experiment about static and dynamic type systems doubts about the positive impact of static type systems on development time,” Paper presented at the, vol. 45, no. 10, pp. 22-35, Oct. 1, 2010.

[2] I. R. Harlin, H. Washizaki, and Y. Fukazawa, “Impact of using a statictype system in computer programming,” Paper presented at the, pp. 116-119, 2017.

[3] B. C. Pierce, “Types and Programming Languages,” 1st ed., Cambridge: MIT Press, 2002. Retrieved from https://ieeexplore.ieee.org/servlet/ opac?bknumber=6267321.