IAES Journal of Electrical and Electronics Engineering
Vol. 12, No. 1, June 2025, pp. 13~17
ISSN: 2338-7432, DOI: 10.11591/jeee.v12i1.pp13-17

Programming languages for artificial intelligence and machine learning: a comparative review of performance, libraries, and development ecosystems

Laksana Talenta Ahmad
Embedded Systems and Power Electronics Research Group, Yogyakarta, Indonesia

Article Info

Article history:

Received Feb 1, 2023
Revised Sep 20, 2023
Accepted Mar 24, 2024

Keywords:

Artificial intelligence
Computational performance
Deep learning
Interoperability
Machine learning
Programming languages
Software libraries

ABSTRACT

The selection of a programming language is an important consideration in artificial intelligence (AI) and machine learning (ML), as it influences computational performance, development productivity, software integration, and deployment scalability. Python, R, Julia, C++, and Java offer different capabilities that address the requirements of modern intelligent applications. This review compares these five languages in terms of execution performance, data-processing efficiency, ML libraries, deep learning frameworks, and development ecosystems. Particular attention is given to numerical computation, neural network development, hardware acceleration, distributed processing, and interoperability between programming environments. The analysis indicates that Python provides the most comprehensive AI ecosystem, while R remains valuable for statistical analysis and research-oriented modeling. Julia offers an attractive balance between high-level programming and computational efficiency, whereas C++ provides extensive control over memory and hardware resources. Java is particularly relevant to enterprise applications and distributed computing infrastructures. Rather than identifying a universally superior language, this review evaluates their complementary strengths and limitations. The findings provide practical guidance for selecting programming technologies according to computational requirements, application complexity, and long-term deployment objectives.

This is an open access article under the CC BY-SA license.



Corresponding Author:

Laksana Talenta Ahmad
Embedded Systems and Power Electronics Research Group
Yogyakarta, Indonesia
Email: laksanatadz@gmail.com

1. INTRODUCTION

Artificial intelligence (AI) and machine learning (ML) have become essential technologies for solving complex problems in engineering, healthcare, manufacturing, finance, transportation, and scientific research. Their development increasingly depends on programming environments capable of handling large datasets, computationally intensive algorithms, and sophisticated neural network architectures. However, implementing an intelligent system involves more than selecting an appropriate learning algorithm because the programming language also influences development time, execution efficiency, memory utilization, and deployment flexibility. A language that simplifies model development may introduce computational overhead, whereas a highly optimized language may require additional programming effort. Consequently, selecting an appropriate programming language represents an important technical decision throughout the AI and ML development lifecycle [1]–[4]. The existing programming landscape provides several alternatives with different design philosophies and development priorities. Python has become a widely adopted

environment because of its extensive scientific libraries and integration with major ML frameworks. R provides specialized capabilities for statistical computing, data visualization, and predictive modeling, while Julia was designed to combine high-level programming convenience with efficient numerical execution. C++ offers direct control over memory management, hardware resources, and computational optimization, whereas Java provides a mature development environment for enterprise software and distributed applications [3], [5]–[9]. These differences influence how developers implement algorithms, process data, train neural networks, and integrate intelligent models into operational systems. Despite the availability of numerous programming languages and frameworks, choosing the most appropriate environment remains challenging. Existing publications frequently concentrate on individual libraries, particular algorithms, or specific computational benchmarks rather than evaluating programming languages across the broader AI development process. Furthermore, comparisons based exclusively on execution time may overlook important factors such as library maturity, hardware acceleration, model portability, software maintenance, and developer productivity. The relationship between programming language and framework performance also complicates direct comparisons because many high-level environments execute computationally intensive operations through optimized native libraries. Therefore, a comprehensive comparison is necessary to understand how programming languages and their surrounding ecosystems contribute to practical AI and ML development.

This review addresses these limitations by comparatively examining Python, R, Julia, C++, and Java across five interconnected dimensions: computational performance, data-processing efficiency, ML libraries, deep learning capabilities, and scalability and interoperability. Its objective is to identify the strengths and limitations of each language while examining the influence of software frameworks and development ecosystems on practical implementation. The review also considers how different languages can complement one another within heterogeneous AI systems. By connecting language characteristics with application requirements, this study provides a structured basis for selecting appropriate programming technologies for research, industrial development, and large-scale intelligent applications.

2. COMPUTATIONAL PERFORMANCE AND DATA-PROCESSING EFFICIENCY

Computational performance depends on the interaction between programming language design, compiler technology, numerical libraries, memory management, and hardware architecture. C++ generally provides strong execution efficiency because compiled machine code, explicit memory management, and low-level optimization allow developers to control computational resources. Java uses just-in-time compilation and runtime optimization, which can deliver competitive performance for long-running applications, although garbage collection and runtime management may introduce additional overhead. Python and R offer more convenient high-level programming interfaces, but interpreted execution can become inefficient when numerical operations are implemented through ordinary loops. Nevertheless, optimized libraries allow both languages to perform substantial numerical workloads through compiled native routines [1], [2], [10], [11].

Python demonstrates that high-level programming does not necessarily imply poor numerical performance. NumPy provides efficient multidimensional arrays and vectorized operations, while SciPy supplies numerical algorithms for optimization, integration, linear algebra, and scientific computation [1], [2]. Numba further improves execution efficiency by compiling selected Python functions into machine code through LLVM-based just-in-time compilation [5]. Julia follows a different approach by specializing functions according to argument types and compiling them into efficient native instructions. Its design aims to reduce the traditional separation between rapid prototyping and high-performance implementation [3].

R is particularly effective for statistical data analysis, exploratory modeling, and visualization, supported by a mature collection of specialized packages. However, computationally intensive operations may require additional optimization or integration with compiled languages. Rcpp addresses this limitation by allowing developers to implement performance-critical operations in C++ while retaining access to R data structures and workflows [8], [9]. This arrangement illustrates that language performance should not be evaluated independently of available optimization mechanisms. In practical applications, data representation, algorithm complexity, parallel execution, and library implementation may influence performance more significantly than the language syntax itself.

3. COMPARATIVE ANALYSIS OF PROGRAMMING LANGUAGES AND AI ECOSYSTEMS

Each programming language offers a distinctive combination of libraries, development tools, and application capabilities. Python provides a broad ecosystem covering data preparation, traditional ML, deep learning, and model deployment. R emphasizes statistical modeling and reproducible analytical workflows, while Julia integrates numerical computing with a growing ML environment. C++ is particularly relevant to

performance-sensitive applications, computer vision, and embedded inference, whereas Java supports object-oriented software engineering and integration with enterprise information systems. Table 1 summarizes their principal characteristics and typical application suitability. The comparison indicates that no language dominates every evaluation criterion. Python offers a particularly balanced environment because its libraries support the complete workflow from data preparation to model deployment. R remains attractive when statistical interpretation, analytical reporting, and established statistical procedures are central to the project. Julia provides an alternative for researchers seeking efficient numerical computation without moving immediately to a lower-level language. C++ and Java become increasingly relevant when intelligent models must operate within performance-sensitive software or established enterprise infrastructures.

Table 1. Comparative characteristics of programming languages for AI and ML development

Evaluation criterion	Python	R	Julia	C++	Java
Execution performance	Moderate to high with optimized libraries	Moderate to high with native extensions	High after compilation	Very high	High with JIT optimization
Data processing	NumPy, pandas, SciPy	data. table, dplyr, data.frame	DataFrames.jl, native arrays	Eigen, STL, OpenCV	Collections, Spark APIs
ML ecosystem	Extensive	Strong statistical ML ecosystem	Growing and specialized	Strong algorithm and inference libraries	Mature enterprise-oriented libraries
Deep learning	TensorFlow, PyTorch, Keras	TensorFlow and Keras interfaces	Flux.jl	Caffe, LibTorch, inference libraries	Deeplearning4j
Hardware acceleration	Extensive GPU and accelerator support	Available through integrated frameworks	GPU ecosystem developing	Extensive low-level control	Framework-dependent
Development productivity	High	High for statistical analysis	High for numerical research	Relatively demanding	Strong enterprise tooling
Typical applications	Research, AI services, data science	Statistics, analytics, predictive research	Scientific ML and numerical modeling	Embedded AI and high-performance inference	Enterprise and distributed applications

4. MACHINE LEARNING LIBRARIES AND FRAMEWORK DEVELOPMENT

The availability and maturity of software libraries strongly influence the practical usefulness of a programming language. Python benefits from a comprehensive ecosystem that includes NumPy, SciPy, and scikit-learn, covering numerical computation, preprocessing, classification, regression, clustering, and model evaluation [1], [2], [4]. Its ecosystem also supports specialized libraries for gradient boosting, including XGBoost and LightGBM, which incorporate algorithmic and system-level optimizations for large-scale learning tasks [12], [13]. These resources reduce the need to implement fundamental algorithms from scratch and encourage reusable development workflows. R provides a substantial collection of packages for statistical learning and predictive modeling. The caret package offers a unified interface for model training, preprocessing, parameter tuning, and evaluation, while mlr3 provides an extensible object-oriented framework for constructing ML workflows [6], [7]. These capabilities are particularly useful in research environments where model comparison, statistical interpretation, and reproducibility are important. However, the availability of numerous packages can also create dependency-management challenges, especially when projects combine packages developed under different assumptions or release schedules.

Julia has developed a specialized ML ecosystem that takes advantage of its numerical computing architecture. Flux provides tools for constructing and training neural networks using Julia-native programming structures, while its integration with automatic differentiation and GPU technologies supports more advanced computational applications [14]. Julia's main advantage is the possibility of developing high-level algorithms without necessarily transferring performance-critical components to another language. Nevertheless, its ecosystem is smaller than Python's, and the availability of specialized packages, tutorials, and experienced developers can vary across application domains. C++ supports ML through libraries such as dlib and Caffe, which demonstrate its capabilities in numerical algorithms, computer vision, and neural network implementation [11], [15]. Its strengths include efficient memory management, hardware-level optimization, and integration with existing software systems. Java provides libraries such as Java-ML and Deeplearning4j-based tools, supporting conventional ML and deep learning within Java applications [16]. These ecosystems are particularly useful when AI functions must be integrated into established software products rather than developed exclusively as independent research experiments.

5. DEEP LEARNING AND NEURAL NETWORK DEVELOPMENT

Deep learning introduces additional computational requirements because neural networks involve repeated matrix operations, gradient calculations, memory-intensive data transfers, and iterative optimization. Python has become a prominent development environment for deep learning because its frameworks combine accessible programming interfaces with optimized computational backends. TensorFlow and PyTorch support neural network construction, automatic differentiation, GPU acceleration, and deployment

across different computing environments [10], [17]. The separation between high-level model definitions and optimized execution components allows developers to work productively without manually implementing every low-level operation. Julia provides an alternative approach through Flux, which enables neural network development using native Julia functions and automatic differentiation. This integration is attractive for researchers who require customized mathematical models or want to combine neural networks with scientific simulations [14]. However, the availability of pretrained models, third-party integrations, and production-oriented deployment tools remains an important consideration when comparing Julia with more established deep learning environments. R can also support neural network development through interfaces to TensorFlow and Keras, although these workflows frequently depend on external computational backends. Therefore, R is often more advantageous for analytical modeling and research workflows than for developing highly customized deep learning infrastructure.

C++ and Java offer different approaches to neural network implementation and deployment. C++ supports efficient inference and performance-sensitive applications through libraries such as Caffe and dlib, while also allowing developers to integrate optimized neural network components into existing software [11], [15]. Java-based Deeplearning4j provides deep learning capabilities within the Java ecosystem and has been integrated with Weka to make neural network development more accessible to users of traditional ML tools [16]. These approaches demonstrate that deep learning performance is not determined solely by the language used to define a model. The underlying framework, numerical kernels, hardware support, and deployment architecture are equally important.

6. SCALABILITY, INTEROPERABILITY, AND DEVELOPMENT CHALLENGES

Scalability is an important consideration when AI applications move from experimental datasets to large-scale operational environments. Distributed computing frameworks such as Apache Spark provide processing capabilities that can be accessed through multiple programming interfaces, including Java, Python, and R [18]. Such architectures allow organizations to process large datasets and construct distributed ML workflows without restricting development to a single programming language. However, distributed execution introduces additional considerations involving data serialization, network communication, memory allocation, and synchronization between computing nodes. Interoperability is equally important because AI systems frequently combine libraries and software components developed in different languages. Python may be used for model development, C++ for performance-critical inference, and Java for integration with enterprise applications. Rcpp demonstrates how native language interfaces can connect R and C++ while retaining access to their respective development environments [8], [9]. Similarly, model exchange technologies such as the Open Neural Network Exchange (ONNX) aim to improve portability between supported ML frameworks. Nevertheless, model conversion does not always guarantee identical numerical behavior because operators, preprocessing procedures, and hardware-specific implementations may differ [19].

Several practical challenges remain relevant across all five programming languages. Dependency compatibility, library version changes, hardware-specific optimization, and software maintenance can affect the reproducibility and reliability of AI applications. Framework defects and inconsistent implementation behavior may also complicate the development of deep learning systems, particularly when applications depend on multiple computational backends [20]. Consequently, developers should evaluate not only initial model performance but also testing facilities, documentation, community support, deployment compatibility, and long-term maintenance requirements.

7. DISCUSSION AND FUTURE RESEARCH DIRECTIONS

The comparative findings suggest that programming language selection should be guided by application requirements rather than general assumptions about computational superiority. Python is a strong choice for general AI development because of its extensive libraries, accessible learning environment, and deep learning support. R remains valuable for statistical analysis and predictive research, while Julia offers considerable potential for scientific ML and performance-oriented numerical applications. C++ is particularly appropriate when low-level optimization, memory control, and efficient inference are essential, whereas Java is useful for integrating intelligent functions into enterprise and distributed software infrastructures. These distinctions indicate that a language's practical value depends on the relationship between its computational characteristics and the surrounding development ecosystem.

Future research should investigate more systematic benchmarking approaches that compare programming environments under equivalent hardware, algorithms, dataset sizes, and optimization settings. Such evaluations should distinguish model training time, inference latency, memory consumption, energy requirements, and development complexity rather than relying on a single execution-time measurement. Greater attention should also be given to interoperability standards, portable model representations, and

automated optimization across heterogeneous hardware platforms. Finally, the growing importance of edge AI and resource-constrained devices creates opportunities for hybrid development architectures that combine high-level model development with optimized native execution. These directions may help reduce the gap between experimental AI research and dependable real-world deployment.

8. CONCLUSION

Python, R, Julia, C++, and Java provide complementary capabilities for AI and ML development. Python offers the broadest general-purpose ecosystem, R provides strong statistical and analytical functionality, and Julia combines high-level numerical programming with efficient compiled execution. C++ remains important for performance-sensitive applications, while Java supports enterprise integration and distributed software development. The comparison also demonstrates that practical performance depends on libraries, compilers, hardware acceleration, data structures, and deployment architecture rather than programming language alone. Therefore, the most appropriate development environment should be selected according to computational demands, ecosystem maturity, interoperability requirements, and long-term application objectives. Hybrid approaches that combine the strengths of multiple languages may provide an effective foundation for scalable, maintainable, and high-performance AI systems.

REFERENCES

[1] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, 2020, doi: 10.1038/s41586-020-2649-2.
[2] P. Virtanen *et al.*, "SciPy 1.0: fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, 2020, doi: 10.1038/s41592-019-0686-2.
[3] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM Review*, vol. 59, no. 1, pp. 65–98, 2017, doi: 10.1137/141000671.
[4] F. Pedregosa *et al.*, "Scikit-learn: machine learning in Python," Jun. 2018, doi: 10.5555/1953048.2078195.
[5] S. K. Lam, A. Pitrou, and S. Seibert, "Numba: A LLVM-based Python JIT compiler," *Proceedings of LLVM-HPC 2015: 2nd Workshop on the LLVM Compiler Infrastructure in HPC - Held in conjunction with SC 2015: The International Conference for High Performance Computing, Networking, Storage and Analysis*, vol. 2015-Janua, 2015, doi: 10.1145/2833157.2833162.
[6] M. Kuhn, "Building predictive models in R using the caret package," *Journal of Statistical Software*, vol. 28, no. 5, 2008.
[7] M. Lang *et al.*, "mlr3: a modern object-oriented machine learning framework in R," *Journal of Open Source Software*, vol. 4, no. 44, p. 1903, 2019, doi: 10.21105/joss.01903.
[8] D. Eddelbuettel and R. François, "Rcpp: Seamless R and C++ integration," *Journal of Statistical Software*, vol. 40, no. 8, 2011.
[9] D. Eddelbuettel and J. J. Balamuta, "Extending R with C++: A brief introduction to Rcpp," *American Statistician*, vol. 72, no. 1, pp. 28–36, 2018, doi: 10.1080/00031305.2017.1375990.
[10] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015, doi: 10.1038/nature14539.
[11] D. E. King, "Dlib-ml: A machine learning toolkit," *Journal of Machine Learning Research*, vol. 10, pp. 1755–1758, 2009.
[12] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, vol. 13-17-August-2016, 2016, doi: 10.1145/2939672.2939785.
[13] G. Ke *et al.*, "LightGBM: a highly efficient gradient boosting decision tree," *Advances in Neural Information Processing Systems*, vol. 2017-Decem, pp. 3147–3155, 2017.
[14] M. Innes, "Flux: elegant machine learning with Julia," *Journal of Open Source Software*, vol. 3, no. 25, 2018.
[15] Y. Jia *et al.*, "Caffe: convolutional architecture for fast feature embedding," *MM 2014 - Proceedings of the 2014 ACM Conference on Multimedia*, pp. 675–678, 2014, doi: 10.1145/2647868.2654889.
[16] S. Lang, F. Bravo-Marquez, C. Beckham, M. Hall, and E. Frank, "WekaDeeplearning4j: a deep learning package for Weka based on Deeplearning4j," *Knowledge-Based Systems*, vol. 178, pp. 48–50, 2019, doi: 10.1016/j.knosys.2019.04.013.
[17] A. Paszke *et al.*, "PyTorch: an imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.
[18] M. Zaharia *et al.*, "Apache spark: a unified engine for big data processing," *Communications of the ACM*, vol. 59, no. 11, pp. 56–65, 2016, doi: 10.1145/2934664.
[19] F. Nagata, S. Sakata, R. Abe, K. Watanabe, and M. K. Habib, "Evaluation of interoperability of CNN models between MATLAB and Python environments Using ONNX runtime model," *AI, Computer Science and Robotics Technology*, vol. 3, 2024.
[20] J. Chen, Y. Liang, Q. Shen, J. Jiang, and S. Li, "Toward understanding deep learning framework bugs," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 6, 2023, doi: 10.1145/3587155.

BIOGRAPHIES OF AUTHORS



Laksana Talenta Ahmad is a researcher at the Embedded Systems and Power Electronics Research Group (ESPERG) in Yogyakarta, Indonesia, and a student in the Bachelor of Information Technology (BIT) program, Department of Information Systems (DIS), Kulliyyah of Information and Communication Technology (KICT) at the International Islamic University Malaysia (IIUM) in Kuala Lumpur, Malaysia. His research interests encompass various aspects of information technology, including organizational informatics, database programming, robotics, and artificial intelligence. He can be contacted at email: laksanatadz@gmail.com.