Indiana Journal of Multidisciplinary Research
Abbreviated Tittle- Ind J Multi Res
ISSN *(Online)*- 2583-3820
Journal Homepage Link- https://indianapublications.com/Journals/IJMR
DOI: https://doi.org/10.5281/zenodo.21623153

INDIANA PUBLICATIONS
PRODUCTIVE AND QUALITY RESEARCH

Research Articles

Volume-06|Issue-04|2026

DOSA – ML Pipeline DSL Compiler

Soumya N G[1], Shrisha Kumar[2], Shriya P[3], Surya R Harithsa[4], Swadha Singh[5]

[1]Assistant Professor, Department of Computer Science and Engineering, RNS Institute of Technology, Bangalore, Karnataka, India.
[2,34,5] Undergraduate Student, Department of Computer Science and Engineering, RNS Institute of Technology, Bangalore, Karnataka, India.

Article History
Received: 05.05.2026
Accepted: 22.06.2026
Published: 25.07.2026

Citation
Soumya, N. G., Shrisha, K., Shriya, P., Harithsa, S. R., Singh, S. (2026) DOSA – ML Pipeline DSL Compiler. *Indiana Journal of Multidisciplinary Research, 6*(4), 476-480.

Abstract: In recent years, machine learning (ML) workflows have become larger, more complex, and harder to manage. This has created a clear need for tools that make development more structured, efficient, and accessible. Our project introduces a Domain-Specific Language (DSL) and compiler designed specifically for building and managing ML pipelines. The DSL uses a simple, human-readable syntax to describe the full pipeline—from data preprocessing to model training, evaluation, and deployment—without requiring the user to handle the low-level details. The compiler then translates these high-level instructions into optimized, executable code for popular frameworks such as TensorFlow, PyTorch, or Scikit-learn. It also performs syntax and semantic checks, resolves dependencies, and ensures the resulting pipelines are correct and efficient. By reducing boilerplate code and promoting modular, reusable workflows, our approach makes ML development faster and more reliable. Most importantly, it opens the door for domain experts and non-programmers to participate in building powerful machine learning solutions.

Keywords: Machine Learning, Domain-Specific Language, Compiler Design, ML Pipelines, Automation, Code Generation

INTRODUCTION

Constructing machine learning (ML) pipelines is inherently complex, involving multiple stages such as data preprocessing, feature engineering, model training, evaluation, and deployment. Each of these stages requires significant coding effort and coordination across multiple tools and frameworks. As a result, ML workflows often become time-consuming, error-prone, and **difficult** to scale, particularly for users without advanced programming expertise.

Traditional approaches rely heavily on general-purpose programming languages, which tend to produce verbose and difficult-to-maintain codebases. This not only increases development time but also introduces challenges in debugging, reproducibility, and collaboration. Furthermore, many existing tools and frameworks are tightly coupled to specific platforms, limiting flexibility and making it difficult to migrate workflows across different environments.

To address these challenges, this work proposes a Domain-Specific Language (DSL) supported by a dedicated compiler for simplifying ML pipeline development. The DSL provides a high-level, human-readable syntax that allows users to define complete workflows without dealing with low-level implementation details. By abstracting complexity, the approach enables users to focus on problem formulation and pipeline logic rather than coding intricacies.

The motivation behind this work lies in improving accessibility, efficiency, and reproducibility in machine learning development. A key objective is to enable domain experts and non-programmers to participate in building ML solutions through a simplified and intuitive interface. Additionally, the system promotes modularity and reusability by allowing workflows to be defined as structured components with explicit dependencies.

Another important goal is to ensure correctness and efficiency through automated compilation. The proposed compiler performs syntax validation, semantic analysis, and dependency resolution, ensuring that the generated pipelines are both valid and optimized. By translating high-level specifications into executable code for widely used frameworks such as Scikit-learn, TensorFlow, and PyTorch, the system bridges the gap between abstraction and execution.

Overall, the proposed DSL-based approach aims to reduce development complexity, enhance collaboration, and provide a standardized methodology for defining machine learning workflows, thereby addressing key limitations in existing solutions.

MATERIALS AND METHODS

***Corresponding Author:** Shriya P

The proposed system is centered around the design and implementation of a Domain-Specific Language (DSL) and a corresponding compiler for simplifying machine learning (ML) pipeline development. The methodology integrates concepts from machine learning, compiler design, and domain-specific language engineering to provide a structured and **efficient** workflow definition mechanism.

Design of the Domain-Specific Language (DSL)
The DSL is designed to provide a high-level, human-readable syntax for defining complete ML pipelines. It allows users to specify operations such as data loading, preprocessing, model training, evaluation, and prediction in a concise and declarative manner. Unlike general-purpose programming languages, the DSL abstracts low-level implementation details, enabling users to focus on workflow logic and problem objectives. The design follows principles of readability, expressiveness, and modularity, ensuring that both technical users and domain experts can effectively utilize the language.

Compiler Architecture
The DSL is supported by a multi-stage compiler that transforms high-level pipeline specifications into executable code. The compiler consists of the following components:

- **Lexical Analyzer:** Converts the input DSL script into tokens by identifying keywords, operators, and identifiers.
- **Parser:** Constructs a structured representation of the program using grammar rules and validates the syntax of the DSL.
- **Semantic Analyzer:** Ensures correctness by checking data dependencies, validating operations, and detecting logical errors during compilation.
- **Code Generator:** Translates the validated DSL representation into executable code compatible with ML frameworks such as Scikit-learn, TensorFlow, and PyTorch.

Execution Model and Code Generation
The compiler follows a transpilation-based approach, where DSL scripts are converted into high-level programming language code (e.g., Python). This generated code integrates with existing ML libraries, allowing seamless execution without requiring users to manually implement the pipeline. The system ensures that generated workflows are optimized and free from common programming errors.

Integration with Machine Learning Frameworks
The proposed system supports widely used ML frameworks such as Scikit-learn, TensorFlow, and PyTorch. By generating framework-compatible code, the DSL ensures flexibility and portability across different environments. This integration enables users to leverage existing libraries while benefiting from the abstraction provided by the DSL.

Theoretical Foundations
The methodology is grounded in established concepts from compiler design, including lexical analysis, parsing, semantic validation, and code generation. It also incorporates principles from software engineering, such as modularity, abstraction, and reusability. Additionally, the design of the DSL is informed by prior research on ML-specific DSLs, which emphasize usability, safety, and performance optimization.

Design Considerations
Several key design considerations guided the development of the system:

- **Usability:** The DSL is designed to be intuitive and easy to learn, reducing the barrier for non-expert users.
- **Modularity:** Pipeline components are defined with explicit dependencies, enabling reuse and maintainability.
- **Scalability:** The system supports complex workflows and can be extended to additional ML frameworks.
- **Reliability:** Built-in validation mechanisms ensure correctness and reduce runtime errors.

Overall, the proposed methodology provides a structured approach to defining and executing ML pipelines by combining high-level abstraction with automated code generation, thereby improving efficiency, accuracy, and accessibility.

RESULTS

To evaluate the effectiveness of the proposed DSL compiler, a machine learning pipeline was implemented using both the DSL and an equivalent implementation in Python using the Scikit-learn framework. This comparison was conducted to assess improvements in code simplicity, development efficiency, and correctness.

Case Study: Sales Prediction Pipeline
A sales prediction workflow was selected as a representative use case. The pipeline involved loading a dataset, performing data preprocessing by removing null values, splitting the dataset into training and testing sets, training a Random Forest model, and evaluating the model using accuracy and confusion matrix metrics.

Using the proposed DSL, the entire workflow was defined in a concise and human-readable format consisting of only a few lines. In contrast, the equivalent implementation in Python required significantly more lines of code, including explicit library imports, data handling procedures, and model configuration steps.

Code Reduction and Simplicity
The DSL representation reduced the code length substantially compared to the traditional implementation. Tasks that typically require multiple lines of procedural code were expressed in a single declarative statement.

***Corresponding Author:** Shriya P

This reduction in verbosity simplifies development and improves readability, particularly for users with limited programming experience.

Abstraction of Complexity
The DSL abstracts common machine learning operations such as data preprocessing, dataset splitting, and model evaluation. Users are not required to manually handle low-level implementation details, as these are automatically managed by the compiler. This abstraction enhances productivity and reduces the likelihood of coding errors.

Correctness and Execution
The compiler successfully translated the DSL script into executable Python code. The generated code produced valid outputs, including accurate evaluation metrics such as accuracy scores and confusion matrices. This confirms that the system maintains correctness while simplifying the workflow definition process.

Reusability and Flexibility
The modular structure of the DSL allows workflows to be easily reused and adapted for different datasets and machine learning models. Users can modify specific components of the pipeline without rewriting the entire codebase, promoting efficiency and consistency across projects.

Preliminary Observations
Initial results indicate that the DSL-based approach reduces development time and minimizes the potential for syntax and logical errors. Although large-scale benchmarking was not conducted, the observed improvements in usability and **efficiency** demonstrate the practical viability of the system.

Overall, the results highlight the effectiveness of the proposed DSL compiler in **simplifying** machine learning pipeline development while maintaining correctness and flexibility.

DISCUSSION

The proposed DSL-based approach for machine learning pipeline development addresses several limitations observed in traditional programming-based workflows. By introducing a high-level abstraction, the system significantly reduces complexity and improves usability, making it accessible to both experienced developers and domain experts.

Analysis of Existing Approaches
Existing Domain-Specific Languages (DSLs) for machine learning can be broadly categorized based on representation, design, typing systems, and execution models. Some DSLs adopt a textual approach, requiring users to write code using predefined syntax, while others provide graphical interfaces that allow pipelines to be constructed visually. Hybrid approaches combine both methods to enhance flexibility.

In terms of design, DSLs may be internal (embedded within a general-purpose language) or external (independent languages with dedicated compilers). Internal DSLs benefit from the features of the host language, whereas external DSLs offer greater flexibility and customization. The proposed system follows an external DSL approach, enabling a tailored syntax specifically designed for ML workflows.

Typing systems also vary across DSLs. Some languages implement static type checking to detect errors during compilation, thereby improving reliability, while others rely on dynamic typing. The proposed system incorporates validation mechanisms inspired by static analysis techniques to ensure correctness.

Execution models differ between interpreted and compiled approaches. Compiled DSLs, such as the one proposed in this work, translate high-level specifications into executable code, enabling optimization and improved performance. The use of a compiler-based architecture also allows integration with advanced infrastructures and supports scalability.

Key Research Trends
Recent developments in ML pipeline design emphasize safety, reliability, and automation. Techniques such as static type checking and semantic validation are increasingly adopted to detect errors early in the development process. Integration with widely used ML frameworks has also become a priority, ensuring compatibility and ease of adoption.

Another important trend is the use of high-level abstractions and user-friendly interfaces to improve accessibility. DSLs are being designed to support both technical and non-technical users, enabling broader participation in machine learning development. Additionally, performance optimization techniques, including implicit parallelism and **efficient** resource utilization, are gaining attention.

Advantages of the Proposed System
The proposed DSL offers several advantages, including reduced code complexity, improved readability, and enhanced productivity. By abstracting low-level details, it allows users to focus on pipeline logic rather than implementation specifics. The modular structure of the DSL promotes reusability and maintainability, while the compiler ensures correctness through validation and error detection.

Limitations and Challenges
Despite its advantages, the system faces several challenges. Interoperability remains a key concern, as ML ecosystems often involve multiple frameworks and tools. Ensuring seamless integration across diverse

***Corresponding Author:** Shriya P

platforms requires standardized interfaces and extensible design.

Performance overhead is another consideration, particularly when dealing with large-scale pipelines. The compilation process and dependency resolution mechanisms may introduce additional computational costs, which need to be optimized.

User adoption also presents a challenge, as many practitioners are unfamiliar with DSL-based approaches. Providing adequate documentation, training, and user-friendly interfaces is essential to encourage widespread usage.

Security and ethical concerns must also be addressed. Automated code generation introduces potential risks such as data leakage, incorrect transformations, and vulnerabilities in generated pipelines. Ensuring transparency, fairness, and compliance with data protection regulations is critical.

Future Directions

Future research can explore hybrid compilation models that combine static and dynamic techniques to improve performance and flexibility. The integration of AI-driven optimization methods may further enhance code generation and pipeline efficiency.

Developing standardized frameworks for interoperability can facilitate seamless integration with existing ML tools and cloud platforms. Additionally, efforts to improve scalability, reduce computational overhead, and enhance energy efficiency will be important for real-world applications.

Further advancements in security mechanisms and governance models will also be necessary to ensure the safe and ethical use of DSL-based ML pipelines. With continued research and development, the proposed approach has the potential to significantly transform machine learning workflow management.

CONCLUSION

This paper presented a Domain-Specific Language (DSL) and a corresponding compiler designed to simplify the development and execution of machine learning pipelines. By introducing a high-level, human-readable syntax, the proposed system enables users to define complete ML workflows without dealing with low-level implementation complexities.

The approach effectively reduces code verbosity, improves readability, and enhances productivity by automating key stages such as preprocessing, model training, and evaluation. Through compiler-based translation, the DSL ensures correctness, reliability, and seamless integration with widely used machine learning frameworks such as Scikit-learn, TensorFlow, and PyTorch.

The results demonstrate that the proposed system significantly reduces development effort while maintaining accuracy and flexibility. The modular design of the DSL promotes reusability and supports efficient workflow management, making it suitable for both academic and practical applications.

Despite its advantages, challenges such as interoperability, performance optimization, and user adoption remain areas for further improvement. Future work will focus on enhancing compiler efficiency, expanding support for additional frameworks, and incorporating intelligent optimization techniques.

In conclusion, the proposed DSL-based ML pipeline compiler provides a structured, efficient, and scalable solution for modern machine learning development, with the potential to bridge the gap between domain expertise and technical implementation.

CONFLICT OF INTEREST

The authors declare that there are no financial or personal conflicts of interest that could have influenced the work reported in this paper.

ACKNOWLEDGEMENTS

The authors would like to express their sincere gratitude to Mrs. Soumya N G, Assistant Professor, Department of Computer Science, RNS Institute of Technology, Bangalore, for her invaluable guidance, continuous support, and encouragement throughout the development of this project.

The authors also extend their appreciation to the Department of Computer Science and Engineering, RNS Institute of Technology, Bangalore, for providing the necessary resources and academic environment to carry out this work successfully.

REFERENCES

1. Van Deursen, A., Klint, P., & Visser, J. (2000). Domain-specific languages: An annotated bibliography. *ACM SIGPLAN Notices, 35*(6), 26–36. https://doi.org/10.1145/352029.352035
2. Kreuzberger, D., Kuhl, N., & Hirschl, S. (2022). *Machine learning operations (MLOps): Overview, definition, and architecture* (arXiv:2205.02302). arXiv. https://arxiv.org/abs/2205.02302
3. Fuertes, A. B., Perez, M., & Meza, J. (2023). Transpiler-based architecture design model for back-end layers in software development. *Applied Sciences, 13*(21).
4. Ovidi, S. O. (2024). Exploring the synergy between programming languages and artificial intelligence: Future trends, challenges and innovations. *International Journal of Innovative Science and Research Technology, 9*(12).
5. Giner-Miguelez, J., Gomez, A., & Cabot, J. (2023). A domain-specific language for describing machine

***Corresponding Author:** Shriya P

learning datasets. *Journal of Computer Languages, 76*, 101209. https://doi.org/10.1016/j.cola.2023.101209

6. Khamis, M. A., & Gomaa, W. (2016). *Domain-specific languages for machine learning.*
7. Harris, H. D. (2018). An architecture and domain-specific language framework for repeated domain-specific predictive modeling. *Proceedings of Machine Learning Research, 82*, 23–32.
8. Reimann, L., & Kniesel-Wünsche, G. (2023). Safe-DS: A domain-specific language to make data science safe. In *Proceedings of the 45th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)* (pp. 72–77).
9. Portugal, I., Alencar, P., & Cowen, D. (2016). A survey on domain-specific languages for machine learning in big data. In *Computation in Big Data Era*. Springer.
10. Oaks, B. J., Famelis, M., & Sahraoui, H. (2022). *Building domain-specific machine learning workflows: A conceptual framework for the state-of-the-practice* (arXiv:2203.06838). arXiv.
11. Xin, D., Wu, E. Y., Lee, D. J.-L., Salehi, N., & Parameswaran, A. (2021). Whither AutoML? Understanding the role of automation in machine learning workflows. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (pp. 1–16).
12. Jahic, B., Guelfi, N., & Ries, B. (2023). SEMKIS-DSL: A domain-specific language to support requirements engineering of datasets and neural network recognition. *Information, 14*(4), 213. https://doi.org/10.3390/info14040213
13. De Conto, E., Genest, B., & Easwaran, A. (2024). Function+Data Flow: A framework to specify machine learning pipelines for digital twinning. In *Proceedings of the ACM International Conference on AI-Powered Software* (pp. 1–9).
14. Johren, F., & Seebacher, S. (2019). Challenges in the deployment and operation of machine learning in practice. In *Proceedings of the European Conference on Information Systems* (pp. 1–17).
15. Sujeeth, A. K., Lee, H., Brown, K. J., Chafi, H., Wu, M., Atreya, A. R., Olukotun, K., Rompf, T., & Odersky, M. (2011). OptiML: An implicitly parallel domain-specific language for machine learning. In *Proceedings of the 28th International Conference on Machine Learning* (pp. 609–616).
16. Lattner, C., Amini, M., Bondhugula, U., Cohen, A., Davis, A., Pienaar, J., Riddle, R., Shpeisman, T., Vasilache, N., & Zinenko, O. (2020). *MLIR: A compiler infrastructure for the end of Moore's law* (arXiv:2002.11054). arXiv.
17. Kourouklidis, P., Kolovos, D., Noppen, J., & Matragkas, N. (2023). A domain-specific language for monitoring ML model performance. In *Proceedings of the ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion* (pp. 266–275).
18. Gonzalez, T., & Díaz-Herrera, J. (2014). *Computing handbook: Computer science and software engineering* (3rd ed.). CRC Press.
19. Cooper, K. D., & Torczon, L. (2011). *Engineering a compiler* (2nd ed.). Morgan Kaufmann.

***Corresponding Author:** Shriya P