Available online to ejournal.pnc.ac.id

Journal of Innovation Information Technology and Application

Journal Page is available to https://ejournal.pnc.ac.id/index.php/jinita

Programming Languages Prediction from Stack Overflow Questions Using Deep Learning

Razowana Khan Mim [1], *Tapu Biswas* [2*]

[1] *Department of Computer Science and Engineering, Daffodil International University, Dhaka, Bangladesh*
[2] *Department of Computer Science, American International University Bangladesh, Dhaka, Bangladesh*

email:[1] *razowanamim5671@gmail.com* , [2]*tapubiswas731@gmail.com*

ARTICLE INFO

Article history:
Received 09 July 2024
Revised 13 October 2024
Accepted 12 December 2024
Available online 30 December 2024

Keywords:
Stack Overflow
Programming language
Deep learning
Question tag
Prediction

IEEE style in citing this article:
R. K. Mim and T. Biswas , "Programming Languages Prediction from Stack Overflow Questions Using Deep Learning," Journal of Innovation Information Technology and Application (JINITA), vol. 6, no. 2, pp. 153–162, Dec. 2024.

ABSTRACT

Understanding programming languages is vital in the ever-evolving world of software development. With constant updates and the emergence of new languages, staying informed is essential for any programmer. Additionally, utilizing a tagging system for data storage is a widely accepted practice. Our study selected queries from a Stack Overflow dataset using random sampling. Then, the tags were cleaned, and the data was separated into title, title + body, and body. After preprocessing, tokenizing, and padding the data, randomly split it into training and testing datasets. Then, various deep learning models were applied, such as Long Short-Term Memory, Bidirectional Long Short-Term Memory, Multilayer Perceptron, Convolutional Neural Network, Feedforward Neural Network, Gated Recurrent Unit, Recurrent Neural Network, Artificial Neural Network algorithms to the dataset to identify the programming languages from the tags. This study aims to assist in identifying the programming language from the question tags, which can help programmers better understand the problem or make it easier to understand other programming languages.

1. INTRODUCTION

The capacity to reliably predict programming languages from textual input can significantly improve various applications in software development, from automating code classification to enhancing developer recommendation systems. This study utilizes a dataset from Stack Overflow, a well-known online programming community, to assess how well several cutting-edge deep learning (DL) models predict programming languages. The ability to program has become vital in many industries in the current digital era. Knowing programming is important in creating software, creating websites, or working on data analysis tasks [1].

The biggest challenge in this field is the complexity of searching programming languages that often contain contextual phrases, code, and language descriptions. However, noise and the irregularity of calculating the final result associated with a dataset of this scale [2]. The programming language models have to be dynamic and expansive because programming languages may evolve, and new ones are constantly being developed. Because the current prediction models must remain useful for different types of queries and to guarantee that they are properly functional, it is crucial to overcome these problems. By building models of programming language prediction into development environments, the tools can suggest the best programming languages to use. It has the advantage of preventing common mistakes and reducing the time it takes to begin a project. Furthermore, by providing tools that are more consistent regarding language detection, the study can improve the cooperation between the developers. This may mean the

*) Corresponding Author : tapubiswas731@gmail.com

creation of better, less ambiguous, and integrated code and less of a problem with defects or mistakes due to minor linguistic differences.

Many DL algorithms afford a rather comprehensive analysis of the programming language prediction, improving both reliability and robustness for a broad spectrum of input patterns. There is a chance to increase the effectiveness and efficacy of beginning and experienced developers. DL models that predict the programming languages are quite wide [3]. DL technologies have revolutionized Natural Language Processing (NLP), mainly in programming languages. In this study, eight DL architectures were used as Long Short-Term Memory (LSTM), Bidirectional Long Short-Term Memory (BiLSTM), Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), Feedforward Neural Network (FNN), Gated Recurrent Unit (GRU), Recurrent Neural Network (RNN), Artificial Neural Network (ANN) to predict programming languages connected with SO queries. These models learn Meaningful representations from sequential data, extracting relevant features from question titles and bodies for a nuanced understanding of programming language context and improving predictions in context-dependent language choices. The aim is to streamline language identification and offer insights into the programming languages driving software development. The predictive power of these models will improve effectiveness in recognizing relevant languages, helping developers resolve coding problems. This study contributes to the software development life cycle by providing a robust tool for developers and researchers. Each of the models applied in the present research is useful in some way to address these complexities. For instance, while CNNs are highly effective for detecting local features in the text [4] because their filters slide locally across the input, LSTM and GRU-based recurrent models are very good at capturing sequential dependence because their states are only updated sequentially [5]. Also, these models offer dissimilar ways to handle the difficulties of textual data in the context of computer languages, and each one pays superior abilities to the field of NLP.

Finally, this research contributes to future work examining other software engineering areas where DL models can be applied. Similar models can be developed to classify different software development frameworks, libraries, or design patterns in addition to language prediction. These program models will be increasingly valuable as software development workflows evolve and progress to pick the best collection of creation tools, frameworks, and approaches. This work has influenced the direction of further developments by demonstrating that DL can contribute to the evolution of the intelligent development environment's part. The prediction models also provide distinct approaches to addressing the challenges of textual data within the domain of computer languages.

2. LITERATURE REVIEW

This section briefly analyses all the previous and present work in this sector.

Isun Chehreh et al. [6] present an approach to an automated tagging system for SO using DL and NLP techniques. The system improves tags' performance by achieving high accuracy and speed, employing a two-step tag extraction and embedding process consisting of the YAKE algorithm and MPNET. The proposed method achieves an accuracy of 3.4 % more precise than the current benchmarks, and therefore, the given method can potentially contribute to the development of tag-based categorization in Q&A platforms.

Artyom Lobanov et al. [7] explored in the work which information source is more appropriate. Valuable for tag prediction. They contrast the present research to compare the existing approaches of each type of network on the same operational dataset and with the same set of tags. Then, they suggest an effective approach, such as an ensemble of the Gated Graph Neural Network model for a press release. At the same time, the model is used in this work to describe the Gated Graph Neural Network model for combining results and Bidirectional Encoder Representations used from the Transformers model for analyzing propositions. The proposed approach is better than that of earlier work. Proposed models by 0:175 of the PR-AUC metric.

Erjon Skenderi et al. [8] evaluated several text representation strategies for applying tag prediction in questions submitted in SO. They also discovered that Sentence-RoBERTa as a text representation method performs better than other text representation methods in successfully identifying the cases that fall in the total percentage of 17 or higher. This enhanced the prediction of tags for queries that do not contain code symbols.

Avigit K. Saha et al. [9] gathered data from millions of questions on the Q&A site SO. Utilizing a discriminative model approach, the system autonomously suggests question tags to guide questioners in selecting pertinent tags for eliciting a response.

Smrithi Rekha V et al. [10] introduced a hybrid auto-tagging system for SO. This system comprises a programming language detection system and an SVM-based question classification system. Upon a user entering a question, the system will provide tag suggestions.

Virik Jain and Jash Lodhavia [11] focus on developing an autonomous tagging system using machine learning (ML) methods such as K-Nearest Neighbor (KNN) and Random Forest (RF), along with important data preprocessing steps like stemming, tokenization, and removing stop words. The research dataset is obtained from kaggle.com, which provides a 10% SO question dataset. RF achieved an average accuracy of 70% across all the tags, while KNN performed slightly better with an accuracy of 75%.

Jyotiska Nath Khasnabish et al. [12] introduced Bayesian learning models to accurately discern the programming language from a written source code. Utilizing 20000 source code files spanning 10 programming languages, the Naive Bayes (NB), Bayesian Network, and Multinomial Naive Bayes classifier models are used to conduct a comprehensive performance comparison to determine classification accuracy on the test data.

Eray Mert Kavuk and Ayse Tosun [13] proposed one-against-all models for the 15 most popular tags and a combined multi-tag classifier to identify the top K tags for a single post. Trained three algorithms to determine how well a post fits a specific tag. The probabilities of a post belonging to each tag are then combined to produce the results of the multi-tag classifier using the best-performing algorithm. Compared the performance of our multi-tag classifier with a baseline approach KNN and found that it achieves 55% recall and 39% F1 score.

Taniya Saini and Sachin Tripathi [14] highlighted a system that efficiently gathers a substantial amount of data from a website and utilizes various methods to accurately predict tags for SO posts and achieve better accuracy for the 1000 most frequent tags.

Srinivas Subramani et al. [15] proposed a tag prediction system compared to MLP and GRU. The system is evaluated based on test accuracy, hamming loss, subset accuracy, Jaccard score, precision, recall, and f1-score.

Juan F. Baquero et al. [16] proposed a method that generates word embeddings representing each question term in a vector space. This allows for operations like comparing words, sentences, and questions. The method was tested on 18,000 questions related to 18 different programming languages. The results demonstrate that extracting valuable, non-obvious information from this highly unstructured data source is possible.

Haoyu Wang et al. [17] evaluated SOTagRec on SO and compared it with state-of-the-art methods; the experimental results show that SOTagRec achieves 81.7% for Recall@5 and 88.7% for Recall@10, outperforming previous relevant methods.

Taniya Saini and Sachin Tripathi developed [18] a system that can gather a large amount of data from a website and utilize this data to predict tags for SO posts from the 1000 used tags, aiming to improve accuracy.

Purvi Prajapati et al. [19] focus on multi-label classification methods and evolutionary measures for multi-label classification. Conducted a comparative analysis of multi-label classification methods based on theoretical study and simulation of various datasets.

This study used eight DL architectures to predict programming languages from SO queries, including LSTM, BiLSTM, MLP, CNN, FNN, GRU, RNN, and ANN. These models extract meaningful features from question data to enhance language identification and offer insights into software development. They provide diverse approaches for handling textual data in the context of computer languages. Some papers used a limited number of questions on different programming languages. Many papers could only provide an accuracy of 70-75 %. Our model used 15 tags with an excellent accuracy between 81-87%. Also, our study helps to choose which model is better for a specific language. The model that can give better accuracy in this field is also clarified.

3. METHODOLOGY

Here, a sequential, step-by-step demonstration of this experiment was conducted. The whole process shows how the experiment maintains a flow to identify the programming language from the tags.

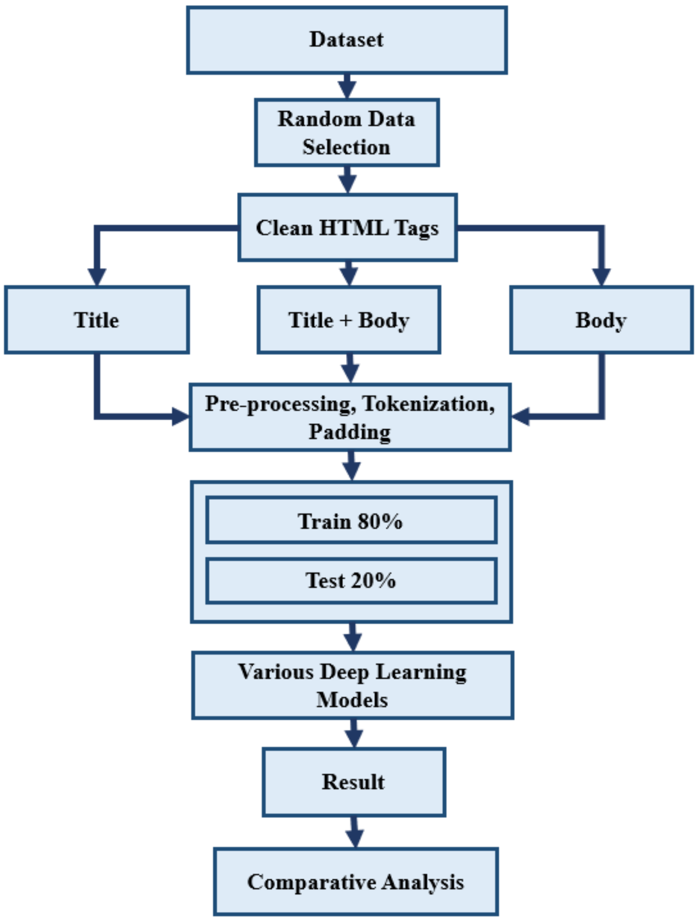


Figure 1. Experimental Workflow

In Figure 1, the flowchart describes how DL models predict tags. First, a random subset of the SO dataset was selected for this task. Then, HTML tags were removed; the raw data was divided into three sections: the title, the body, and a mixture of the two. To standardize the input, these stages were flowed: preprocessing the text, tokenization, and padding. The dataset was split into a training (80%) and testing (20%) set. Then, the processed data was fitted into various DL models for classification to predict the target tags. Finally, the best-performing model is identified by comparing and analyzing the output from these models. Further details are discussed in the Data Collection Procedure, Data Processing, and Result & Discussion.

3.1. Data Collection Procedure

The dataset was collected from Kaggle. This dataset contains the questions and responses from the programming Q&A section of the SO website. Three files are coordinated for this. The questions file has the following information: title, body, creation date, closure date, score, and owner ID. For each question, the answers provide the owner ID, score, body, and creation date. The Inquiries table is rejoined by the ParentId section. The tags file includes the Tags and ID for each of these questions. More than two million pieces of data are in this dataset. There are 3 files, but the Question.csv and Tags.csv files were utilized. The question.csv file was added with the tag.csv file. There are 3750994 data in the tag.csv file and about 1264216 data in the question.csv file. The most popular 15 programming language tags from the entire dataset to perform better. Java, javascript, jQuery, MySQL, objective-c, PHP, Python, android, asp.net, C#, C++, CSS, HTML, iOS, and SQL are those classes.

3.2. Data Processing

An important first step in getting the dataset ready for ML model training is data preparation. The data is prepared for model training through preprocessing procedures. Before providing the input data (title, body) and target variable (tags) to the models for training, they needed to be suitably formatted and tokenized. The tag file and the question file were added using the ID column as a guide. Tokenization and padding of the question bodies and titles are part of the data preprocessing for this study. The raw text input is transformed into sequences of integers using the tokenizer class, where each separate word is given its integer index. Each SO question's title and body go through this process independently. The texts_to_sequences method is then applied to the training data to generate the sequences_train_title and sequences_train_body. Padding is applied to ensure reliable length using the sequence. the pad_sequences method. Then, the whole dataset was divided into training and testing datasets. The title, body, and concatenation of all eight models are trained using these processed sequences as inputs.

3.3. DL Algorithms

The deep learning algorithm can train itself from errors, so it was suitable for identifying programming language from SO question tags.

LSTM: LSTM networks use input, forget, output, and memory gates to capture long-term dependencies in sequential data. These gates regulate information flow, allowing LSTM to retain and update cell states across extended sequences. Applications include face recognition, handwriting recognition, and time series prediction, with potential real-world use in business, health, and transportation [20].

BILSTM: BILSTM process sequences in both forward and backward directions, capturing dependencies from past and future contexts. They combine forward and backwards hidden states to capture complete context information. BILSTM neural networks, or BILSTM, are employed in the paper's suggested attention-based short-term wind power prediction model [21].

MLP: MLPs are ANNs with multiple layers that study complex functions by joining linear transformations with non-linear activation functions. They also stack some entirely connected layers to model complex relationships between inputs and outputs. MLPs are an essential tool in DL theory because they improve performance with scale, which challenges the significance of inductive bias [22].

CNN: CNN is commonly used for processing grid data structures like images. They automatically learn spatial hierarchies of features and have proven effective in tasks such as text categorization and pattern recognition. Extensive analysis and testing confirm their suitability for text [23].

FNN: FNN is a type of neural network where node connections don't form a cycle. They process input data through successive layers without considering temporal dependencies. In previous years, some areas have shown a strong interest in FNN optimization by researchers and practitioners [24].

GRU: The GRU is a kind of RNN that uses update and reset gates to alleviate the disappearing gradient problems. It efficiently captures dependencies in sequential data. The update gate vector, reset gate vector, candidate activation, and final memory at time t are included in the update and reset gates formulas. The update gate vector is a sigmoid activation function, the reset gate vector is a weight matrix, and the candidate activation is a hyperbolic tangent activation function. The final memory at a time is the final hidden state at the time step. GRU selectively updates and resets the hidden state using the gates, thereby efficiently learning dependencies in sequential data [25].

RNN: RNN recognizes patterns in sequential data, maintaining hidden states to capture temporal dependencies. It also uses recurrent connections to maintain and update these hidden states across time steps, enabling the modelling of sequential data [26].

ANN: ANN is the foundational model of DL, consisting of interconnected neurons arranged in layers. They transform input data through multiple layers to produce a final output, using activation functions to introduce non-linearity. The output of the hidden layer neuron is represented by h(h)j. The output is a weight matrix and bias term [27].

3.4. Performance Metrics

Accuracy: The accuracy of a classification model in making predictions over time is measured by comparing the total correct predictions to the sum of false negatives and true positives. However, accuracy may not be suitable for imbalanced datasets with unequal class representation.

$$Accuracy = \frac{TP + TN}{TP+FP+FN+TN} \quad (1)$$

Recall: The recall of a classification model assesses a classifier's performance in class imbalances by giving an additional detailed assessment of how well it predicts positive examples. Model recognition is also assessed by the recall.

$$Recall = \frac{TP}{TP+FN} \quad (2)$$

Precision: The most important metrics for classification jobs are precision and recall, mainly when dealing with unbalanced datasets. Calculating the true positive rate, the ratio of true positives to false negative precision evaluates the accuracy of positive forecasts.

$$Precision = \frac{TP}{TP+FP} \quad (3)$$

F1-score: The main assessment statistic employed is the F1 score. This results from macro F1 assigning an equal weight to each class. The F1 score can originate for a given class using the formula below.

$$F1 - score = \frac{TN+TP}{2TP+FN+FP} \quad (4)$$

4. RESULT & DISCUSSION

The results, which show the accuracy, recall, and precision, as well as the F1 score, of various DL models, such as LSTM, BiLSTM, MLP, CNN, FNN, GRU, RNN, and ANN, explicate some of the ways in which the models can differ in conception.

As seen in Table 1, the CNN and BiLSTM models are the overall best. For this dataset, which evaluated CNN model performance, the accuracy, recall, precision, and F1 scores were 87%. In the same way, the BiLSTM model performed 86% F1 score, the same result as CNN for accuracy and 87% for both recall and precision scores. Also, excellent results were obtained for both the MLP and ANN models; all of the metrics yielded 85% accuracy, which proves that both models have a high ability to train on patterns found in the dataset.

On the other hand, LSTM was relatively lower than MLP and ANN but still reasonably well, scoring an average of 84% in all metrics. The FNN and GRU models are very similar. FNN has a slight decrease in the F1 score, which equals 82 %, while all the other indicators are around 83 %. The same can be said of the GRU model, where recall and accuracy are equal to 82%, and precision is 83 %.

The RNN model reported the lowest overall results, with overall accuracy, recall, precision, and F1 around the 81-80% mark. This implies that compared to LSTM, BiLSTM, and CNN, with numerous hidden layers and more complex architecture, RNN may fail to manage elaborate data representations or perform long-term dependency tests for this dataset.

BiLSTM and CNN showcased the highest accuracy, 87%, among all the prediction models. RNN displays poorer results than other prediction models. BiLSTM and CNN can be used to predict the programming language from SO. It will showcase excellent results in this field. Models like RNN, which has the lowest performance, may require modifications or may not be as well-suited for this specific task. Along with the accuracy, the recall, precision, and f1 score for various algorithms have been demonstrated in Table 1.

Table 1. The Result of DL Models Prediction

Model	Accuracy	Recall	Precision	F1 Score
LSTM	84	84	84	84
BILSTM	87	87	87	86
MLP	85	85	85	85
CNN	87	87	87	87
FNN	83	83	83	82
GRU	82	82	83	82
RNN	81	81	81	80
ANN	85	85	85	85

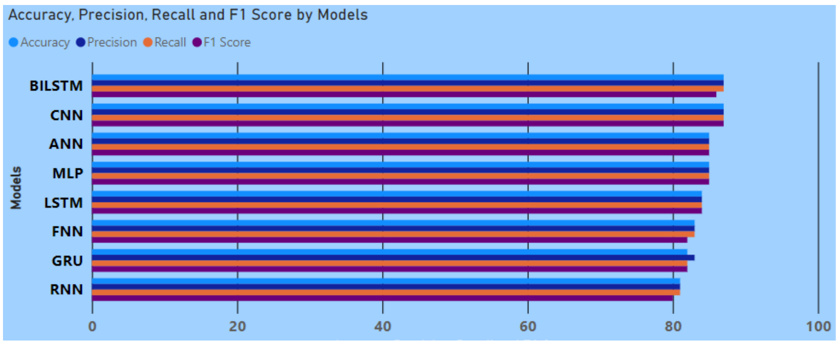
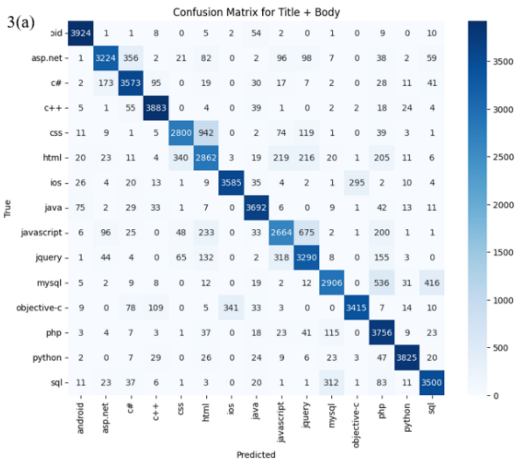


Figure 2. Demonstration Of Prediction Model Result

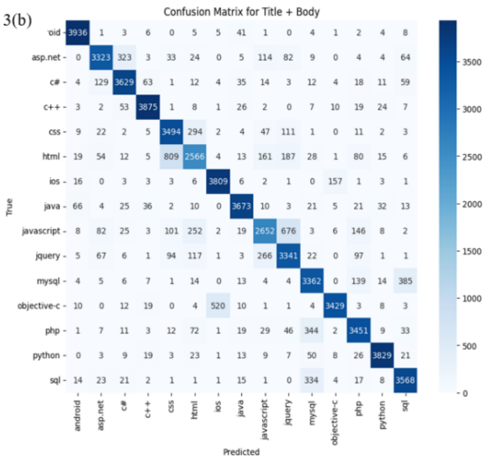
Figure 2 displays the Accuracy, Precision, Recall, and F1 Score of the different models, with BiLSTM outperforming the others by a small margin.

3(a)



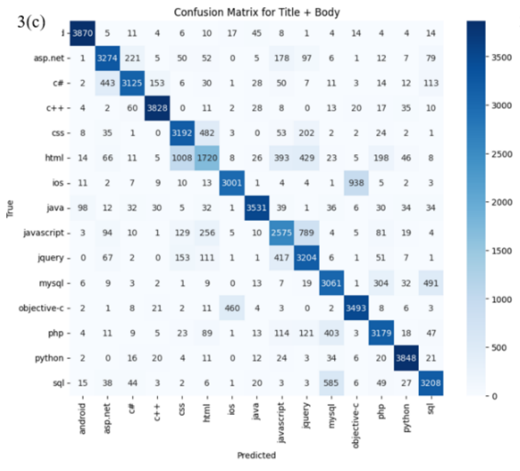
ANN

3(b)



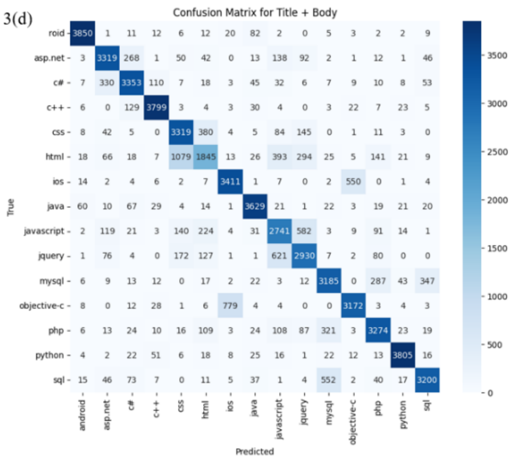
BiLSTM

3(c)



CNN

3(d)



FNN

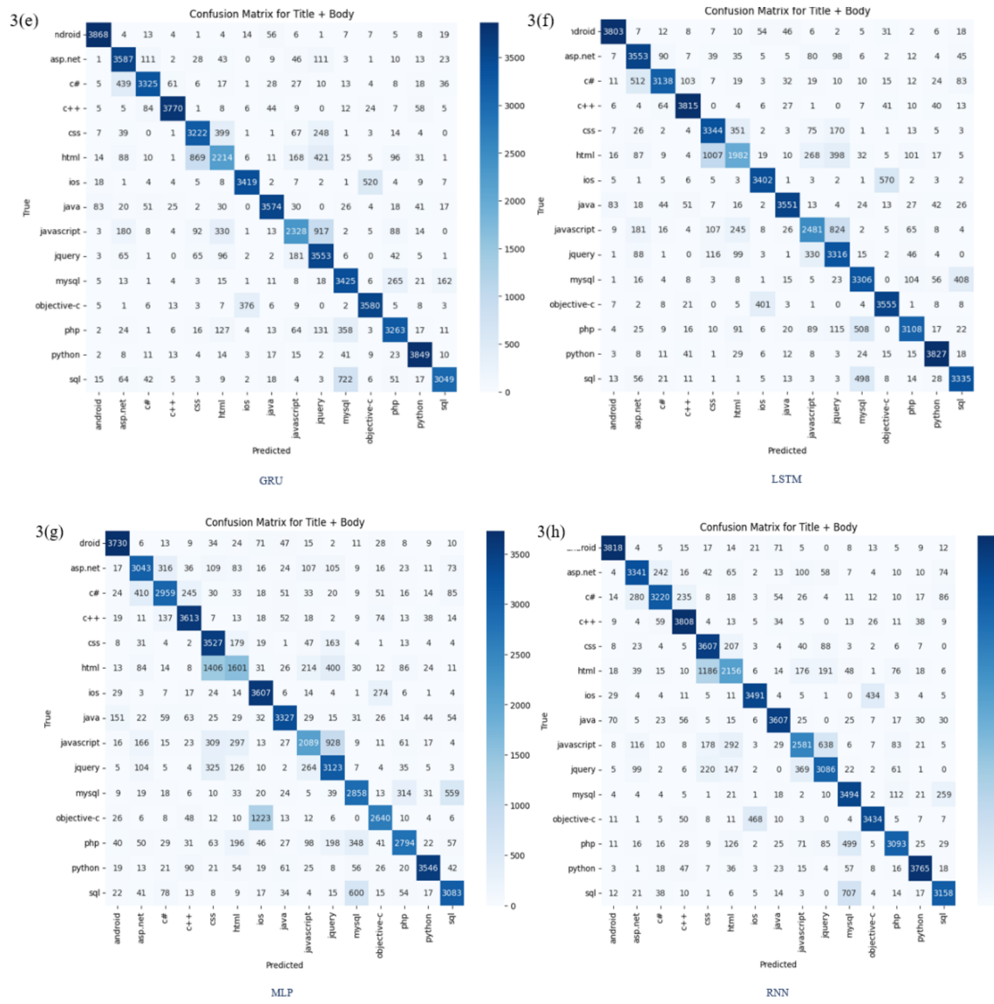


Figure 3. Confusion Matrix of all Models

Figure 3(a) considers strong diagonal elements in the ANN model, especially with the programming languages: “android”, “c#”, “java”, and “python”. A good classification accuracy is proposed. Nevertheless, there are some errors with classification, namely between “javascript” and “jquery” as well as “c#” and “c++”. In Figure 3(b) BiLSTM model has low loss in the diagonal and a comparatively higher attentiveness on small correct predictions of more than two classes, including “android”, “C#”, “java”, and “python”. The more focused diagonal elements show that BiLSTM does slightly better than the other DL models in the least misclassification scenario. In Figure 3(c) CNN model also showed promising results in classes such as ‘android’, ‘c#’, ‘java’, and ‘python’, as highlighted by the diagonal cells in black, which indicate several correct classifications. Nevertheless, there are a few notable misclassifications: for instance, ‘javascript’ is confused with ‘jquery’, and ‘c++’ is confused with ‘c#’. Figure 3(d) showcases similar strong diagonal elements in the FNN Model, which implies a good performance for languages such as Java, Python, android, and c#. Figure 3(e) considers the GRU matrix's results; the model behaves reasonably with some breakthroughs in such tags as ‘android’, ‘c++’, ‘java’, ‘python’, and ‘SQL’. Figure 3(f) has quite similar high prediction values in” android”, “c++”, “java,” and “python” The LSTM model performs as well as the other one; however, the distribution of the prediction model is not the same as the GRU’s one. In Figure 3(g), major classes such as “python”, “SQL”, and “java” were predicted to be good in the MLP model; a diagonally shaded value represented good prediction. However, there are some conflicts between languages like c++ and c# and between web languages like HTML and CSS. In Figure 3(h), the accuracy of the RNN model is quite close to the MLP model in the classes like “python”, “SQL”, and “java”. This is confirmed by a high prediction score on the “android”, “java”, and “objective-c” diagonals.

5. CONCLUSION

Programming languages are very important for conveying instructions to computers. The generation of programs and the formulation of algorithms are constantly developing to continue with technical developments [28]. The SO dataset contains noisy and irrelevant data from incomplete questions, poorly formatted text, and irrelevant information. This leads to information loss during preprocessing, reducing the effectiveness of models by missing critical contextual information or being influenced by irrelevant content. In this paper on categorizing programming languages by DL from SO questions CNN and BiLSTM were efficient, having 87% accuracy, recall, precision, and F1-measure. This was because of their suitability in identifying advanced patterns in textual data that proved ideal for this task. As for MLP and ANN also achieved 85% across all the metrics, although their more basic structure raised them just a little. LSTM was next in line with RNN, and GRU was slightly behind, where RNN recorded the lowest accuracy. These results have shown the improved performance of CNN and BiLSTM in language point prediction, and future work can refine both models to obtain better results. Our model fails to showcase a novel or hybrid system to predict the accuracy. The model is a detection model, not a prevention model.

For further work, consider sophisticated model architectures and adjusting hyperparameters to enhance the precision of the prediction is our goal. Such models could be incorporated into a cloud-based application where the user posts questions and gets a solution as soon as possible. In addition, future research opportunities could stretch from SO prediction to automatic error prevention and self-healing applications. Such systems would look for call stacks analyzing their states in real-time to notice when memory has given in beforehand. Recording such faults that may include SOs was necessary since it could be used to make appropriate changes to the frequency of function calls depending on the memory resource available. These innovations would increase software reliability and minimize possible mistakes while going further than selection through simpler error correction relying solely on self-prediction.

REFERENCES

[1] T. S. Jalolov, “The Importance of English in Programming,” World of Science, vol. 7, no. 5, pp. 128–134, 2024.

[2] O. Chew, H. T. Lin, K. W. Chang, and K. H. Huang, "Understanding and Mitigating Spurious Correlations in Text Classification with Neighborhood Analysis," arXiv preprint arXiv:2305.13654, 2023. doi: https://doi.org/10.48550/arXiv.2305.13654.

[3] A. Hassan and A. Mahmood, "Efficient deep learning model for text classification based on recurrent and convolutional layers," in 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 1108-1113, Dec. 2017. doi: 10.1109/ICMLA.2017.00009.

[4] K. P. Chaithanya and J. G. Melekoodappattu, "An Exploration on Plant Disease Detection," in 2022 Third International Conference on Intelligent Computing Instrumentation and Control Technologies (ICICICT), pp. 911-916, 2022.

[5] S. Nosouhian, F. Nosouhian, and A. K. Khoshouei, "A review of recurrent neural network architecture for sequence learning: Comparison between LSTM and GRU," 2021. doi:10.20944/preprints202107.0252.v1.

[6] I. Chehreh, E. Ansari, and B. S. Bigham, "Advanced Automated Tagging for Stack Overflow: A Multi-Stage Approach Using Deep Learning and NLP Techniques," in 2024 20th CSI International Symposium on Artificial Intelligence and Signal Processing (AISP), pp. 1-6, Feb. 2024. doi: https://doi.org/10.1145/3387940.3391491.

[7] A. Lobanov, E. Bogomolov, Y. Golubev, M. Mirzayanov, and T. Bryksin, "Predicting tags for programming tasks by combining textual and source code data," arXiv preprint arXiv:2301.04597, 2023. doi: 10.48550/arXiv.2301.04597.

[8] E. Skenderi, S. M. Laaksonen, J. Huhtamäki, and K. Stefanidis, "Assessing Text Representation Methods on Tag Prediction Task for StackOverflow," in The 56th Hawaii International Conference on System Sciences, pp. 585-594, Jan. 2023. doi: https://doi.org/10.24251/HICSS.2023.075.

[9] A. K. Saha, R. K. Saha, and K. A. Schneider, “A discriminative model approach for suggesting tags automatically for Stack Overflow questions,” in 2013 10th Working Conference on Mining Software Repositories (MSR), 2013, pp. 73–76. doi:10.1109/MSR.2013.6624009.

[10] V. S. Rekha, N. Divya, and P. S. Bagavathi, “A hybrid auto-tagging system for Stack Overflow forum questions,” in Proc. 2014 Int. Conf. Interdiscip. Adv. Appl. Comput., 2014, pp. 1–5. doi:10.1145/2660859.2660970.

[11] V. Jain and J. Lodhavia, “Automatic question tagging using k-nearest neighbors and random forest,” in 2020 Int. Conf. Intell. Syst. Comput. Vision (ISCV), 2020, pp. 1–4. doi:10.1109/ISCV49265.2020.9204309

[12] J. N. Khasnabish, M. Sodhi, J. Deshmukh, and G. Srinivasaraghavan, “Detecting programming language from source code using Bayesian learning techniques,” in Mach. Learn. Data Mining Pattern Recognit., 10th Int. Conf., MLDM 2014, St. Petersburg, Russia, Jul. 21–24, 2014. Proc., 2014, vol. 10, pp. 513–522. doi: https://doi.org/10.1007/978-3-319-08979-9_39.

[13] E. M. Kavuk and A. Tosun, “Predicting Stack Overflow question tags: A multi-class, multi-label classification,” in Proc. IEEE/ACM 42nd Int. Conf. Softw. Eng. Workshops, 2020, pp. 489–493. doi: https://doi.org/10.1145/3387940.339149.

[14] T. Saini and S. Tripathi, “Predicting tags for Stack Overflow questions using different classifiers,” in 2018 4th Int. Conf. Recent Adv. Inf. Technol. (RAIT), 2018, pp. 1–5. doi: 10.1109/RAIT.2018.8389059.

[15] S. Subramani, S. Rajesh, K. Wankhede, and B. Wukkadada, “Predicting Tags of Stack Overflow Questions: A Deep Learning Approach,” in 2023 Somaiya Int. Conf. Technol. Inf. Manage. (SICTIM), 2023, pp. 64–68. doi: 10.1109/SICTIM56495.2023.10105054.

[16] J. F. Baquero, J. E. Camargo, F. Restrepo-Calle, J. H. Aponte, and F. A. González, “Predicting the programming language: Extracting knowledge from Stack Overflow posts,” in Adv. Comput., 12th Colombian Conf., CCC 2017, Cali, Colombia, Sep. 19–22, 2017, Proc., 2017, vol. 12, pp. 199–210. doi: https://doi.org/10.1007/978-3-319-66562-7_15.

[17] H. Wang, B. Wang, C. Li, L. Xu, J. He, and M. Yang, "SOTagRec: A combined tag recommendation approach for Stack Overflow," in Proc. 2019 4th Int. Conf. Math. Artif. Intell., 2019, pp. 146–152. doi: https://doi.org/10.1145/3325730.332575.
[18] T. Saini and S. Tripathi, "Predicting tags for Stack Overflow questions using different classifiers," in 2018 4th Int. Conf. Recent Adv. Inf. Technol. (RAIT), 2018, pp. 1–5. doi: 10.1109/RAIT.2018.8389059.
[19] P. Prajapati, A. Thakkar, and A. Ganatra, "A survey and current research challenges in multi-label classification methods," Int. J. Soft Comput. Eng. (IJSCE), vol. 2, no. 1, pp. 248–252, 2012.
[20] D. Wihardini, "Long Short-Term Memory (LSTM): Trends and Future Research Potential," Future, vol. 5, p. 6. doi: 10.46338/ijetae0523_04.
[21] Z. Chai, "BiLSTM Short-term Wind Power Prediction Based on Attention Mechanism," in 2023 IEEE 3rd Int. Conf. Electron. Technol., Commun. Inf. (ICETCI), 2023, pp. 1341–1346.
doi: 10.1109/ICETCI57876.2023.10176962.
[22] G. Bachmann, S. Anagnostidis, and T. Hofmann, "Scaling MLPs: A tale of inductive bias," in Adv. Neural Inf. Process. Syst., vol. 36, 2024.
[23] M. Umer, Z. Imtiaz, M. Ahmad, M. Nappi, C. Medaglia, G. S. Choi, and A. Mehmood, "Impact of convolutional neural network and FastText embedding on text classification," Multimed. Tools Appl., vol. 82, no. 4, pp. 5569–5585, 2023. doi: https://doi.org/10.1007/s11042-022-13459-x.
[24] V. K. Ojha, A. Abraham, and V. Snášel, "Metaheuristic design of feedforward neural networks: A review of two decades of research," Eng. Appl. Artif. Intell., vol. 60, pp. 97–116, 2017. doi: https://doi.org/10.1016/j.engappai.2017.01.013.
[25] W. A. Degife and B. S. Lin, "Deep-Learning-Powered GRU Model for Flight Ticket Fare Forecasting," Appl. Sci., vol. 13, no. 10, p. 6032, 2023. doi: https://doi.org/10.3390/app13106032.
[26] W. Liu, F. Teng, X. Fang, Y. Liang, and S. Zhang, "An RNN-based performance identification model for multi-agent containment control systems," Mathematics, vol. 11, no. 12, p. 2760, 2023. doi: https://doi.org/10.3390/math11122760.
[27] X. Ma, L. Zhong, and X. Chen, "Application of Hopfield Neural Network Algorithm in Mathematical Modeling," in 2023 IEEE 12th Int. Conf. Commun. Syst. Netw. Technol. (CSNT), 2023, pp. 591–595. doi: 10.1109/CSNT57126.2023.10134711
[28] S. Sridhar, "A study on various programming languages to keep pace with innovation," IJITR, vol. 5, no. 2, pp. 5681–5704, Feb.–Mar. 2017.