

International Journal of
Engineering Research and Science & Technology



ISSN : 2319-5991

www.ijerst.com

Email: editor@ijerst.com or editor.ijerst@gmail.com

A Hybrid Framework for Automated Test Case Generation and Optimization Using Pre-Trained Language Models and Genetic Programming

¹Naga Sushma Allur

AIA Insurance,
Victoria, Australia

Nagasushmaallur@gmail.com

²R. Hemnath

Kamadhenu Arts and Science College,
Sathya Mangalam, India.

hemnathrmca@gmail.com

Abstract

Automated test case generation plays a pivotal role for software quality, reliability, and efficiency, while traditional methods such as rule-based and random test generation often produce invalid or redundant test cases. With search-based algorithms like Genetic Algorithms (GA) and Particle Swarm Optimization (PSO), however, both approaches run into significant obstructions due to the complexity of software structures, which limits their effectiveness. To solve these problems, this work demonstrates a Hybrid Framework for Automated Test Case Generation and Optimization, combining Pre-Trained Language Models (PLMs) with Genetic Programming (GP). The PLMs generate an assortment of context-aware test cases by analysing software requirements, whereas the GP optimizes test cases in terms of fault detection, coverage, and execution efficiency. The experimental results that exemplify the PLM-GP method show it performing much better than traditional and PLM-only methods, with 92.5% statement coverage, 85.3% branch coverage, and 89.1% rate defect detection, while decreasing redundancy to 12.5%. Along with that, by comparing the performance with PLM-EA, PLM-GP shows further increase out of the achieved ones: higher accuracy of 0.98, higher diversity of 0.91, and higher 0.96 in coverage with an increased fitness score of 0.97. Intelligent test case generation combined with the pre-trained language models' genetic programming optimizes the effectiveness and efficiency of software testing. The study shows that PLM-GP is a powerful and scalable methodology for automated test case generation, improving fault detection, test coverage, and overall quality above the current techniques. Combining or reducing redundancies, as well as optimizing test execution, enhance software reliability while reducing human effort during the testing process.

Keywords: Automated Test Case Generation, Pre-Trained Language Models, Genetic Programming, Software Testing Optimization, Fault Detection.

1. Introduction

With evolving software systems, more innovation is expected for their testing methodology to be more efficient and intelligent to ensure quality, reliability, and performance [1]. Developments in applications have actually taken on such complexity that traditional manual testing is no longer sufficient to cover very large areas in very little time. Automated test case generation and optimization are milestone techniques developed towards attaining full coverage with reduced human effort and increased efficiency in defect detection [2]. Yet, the development of meaningful, optimized, and diverse test cases that effectively uncover hidden defects within the software remains a pressing research challenge.

The currently existing techniques for automated test case generation such as rule-based methods, random testing, and search-based algorithms primarily end up generating invalid and context-unaware test cases. Although search-based techniques such as GA and PSO appear promising, they cannot be applied effectively to complex syntactic and semantic structures and often result in irrelevant and redundant test cases [3]. In addition, some relatively new approaches are however not designed for adaptation into domain-specific language models, which may result in syntactically correct but logically invalid test cases that do affect the whole testing effectiveness and increase the effort to validate the test cases.

To provide a solution to these limitations, this proposes the Hybrid Framework for Automated Test Case Generation and Optimization based on Pre-Trained Language Models (PLMs) for intelligent, context-aware test case generation, while Genetic Programming focuses on optimally refitting these test cases to maximize fault

detection and coverage improvement [4]. The PLMs generate meaningful and assorted test cases based on software requirements, while GP refines the generated cases to achieve maximum effectiveness. The hybrid approach provides a scalable and efficient approach to overcoming the shortcomings of the existing approaches to test generation.

Research Contributions

- Hybrid PLM-GP framework for automated test case generation is being presented for better fault detection and improved test diversity.
- It reinforces reliability by generating different classes of effective test cases that are context-aware and minimally redundant.
- Includes AI techniques to automate test case generation, thus reducing effort and increasing accuracy and efficiency in testing.

2. Literature Survey

Under this proposal, an innovative application of neuromyotonic tensor networks, metaheuristic optimization and social influence-driven reinforcement learning may realise the newly proposed model of adaptable artificial intelligence into software engineering applications [5]. Changes associated with their inclusion into the software or test applications include: AI integration, automated fault injection, real-time adaptive testing, each adding considerable computational overhead and resource costs [6]. This also complicates the management of large fault libraries and considerably increases the overall expense of the cloud infrastructure. Changes, whether minor or major, are sought while developing algorithms for more advanced applications in specialized fields such as robotics or autonomous vehicles [7]. This research paper works through a high-performance machine learning pipeline that carries out feature selection, training, and data representation for ELM and SRC. The primary goal of this research is to improve classification performance, increase model robustness in multidimensional data labelling, and develop a PSO-tuned QDA parameter optimization framework directed towards efficient AI software applications [8]. Strong software testing systems have therefore to be put into place as early as possible to judge and fix the faults hidden by those AI techniques. The advanced Genetic Algorithms (GAs), hybrid algorithms integrating GAs with PSO and ACO, adaptive mechanisms, and co-evolutionary techniques are employed in this study to enhance test data generation, path coverage, and testing efficiency [9]. It makes use of DBSCAN and Constant Channel Rate (CCR) for speed anomaly detection and bandwidth efficiency in this study; however, the existing system may require optimization for handling scalability in a more complex and larger size mobile network environment. In the supplied study, a hybrid deep learning model consisting of a stacked autoencoder which is for dimensionality reduction and coupled with an SVM classifier for phishing detection [10]. Technology is developed for the detected malware but comes with limitations on real-time adaptive learning and scaling in a dynamic environment. A unique load balancing strategy with edge computing technology, AI, and machine learning are introduced in the paper; however, this is found to be disadvantageous with the current approaches, intended for dynamic-cloud environments and mandates paradigm changes in efficient resource allocation [11]. Adapted for the research, AHSEM stands for Adaptive Heterogeneous Structural Equation Modelling networking in strategic business planning for knowledge management. A significant drawback noticed is the poorly defined development plan that leads to the failures. Big data analytics, DSS, and mixed-integer linear programming are applied in the present study for optimization of software testing [12]. It lacks real-time adaptation learning and is in need of advanced machine learning techniques to yield improved defect prediction. It is used for predictive analytics as well as healthcare administration, supported by a framework of AI, Big Data Mining as well IoT. However, this framework does not seem to be ready yet for handling diverse and dynamic healthcare data [13]. SOMs can be used for the clustering of test cases along with defect pattern identification. Thus, they can be used to optimize test suites and enhance fault detection accuracy, compared to traditional ones [14].

In this study, data clustering was effectively accomplished using clustering techniques like NCA, MDS, BIRCH, and LPWAN; in software testing, these techniques are applicable for test case optimization and defect pattern clustering, thereby enhancing testing efficiency for large datasets [15]. In this study, key techniques such as encryption, access control, and user authentication can be adapted in software testing to assess the security and resilience of applications. A study includes adaptation of the Global Authentication Register System (GARS) approach to software testing for evaluating authentication, authorization, and privacy [16]. This study employs the integration of pre-trained language models with evolutionary algorithms for automated test case generation in software testing, which enhances the accuracy, diversity, and coverage of the tests [17]. Here an augmented Post-Marketing Surveillance (PMS) paradigm is embraced in software testing to ensure continuous validation and risk assessment of AI systems. The AI support framework would monitor the quality of data, validate the AI system for decision-making, and ensure ethical and regulatory standards, thereby lending credibility and trustworthiness

to AI-supported software solutions [18]. This hybrid AGSVR-LSTM-HMM framework shall enhance software testing through vulnerabilities. It shall also impart malicious patterns to software for more robust and reliable. In this study, the combined use of ANNs and FEA models can be adapted to perform software testing in simulating and evaluating system behaviours under various conditions, improving strong performance testing, fault prediction, and reliability assessment for complex software-driven systems [19]. The framework developed in this study can be adapted to enhance software testing intended for validation of security, reliability, and compliance in m-health applications concerning their authentication mechanisms and data handling processes. Such a multi-stage framework can also be used for automatic defect detection and classification wherein feature extraction and classification techniques help identify complex patterns of software defects, hence improving accuracy and efficiency in the overall testing process [20]. The hybrid test generation methodology that combines pre-trained language models and evolutionary algorithms automates the generation of effective, diverse, and optimized test cases, thus enhancing software testing. This study shows that the mediation effect models and threshold regression modeling methods from the simulation domain can be applied in software testing to examine the effect of different testing strategies on software quality enhancement [21]. These compute AI-based techniques as well as deep learning algorithms, which would be applicable toward the automation of test case generation, fault prediction, and optimization of test coverage in software testing. In this investigation, XML treatments are examined for test cases, along with injection of faults and cloud testing, but the tasks of updating fault libraries and handling adaptive testing complexity are yet unresolved [22]. Much of the research on hybridization concerned inbreeding of neural networks with some heuristics to improve the software testing aspect, such as test case prioritization, fault detection rates, and code coverage, which may make testing more efficient and effective as a classical TCP alternative [23]. Much of the capital offered by this study can address many issues through the joint application of AI-enabled CAPTCHA, graphical passwords, and encryption techniques to software testing for the validation of authentication mechanisms, security protocols, and the robustness of systems against unauthorized access in high-security applications [24].

3. Problem Statement

This section provides an overview of recent developments related to AI and hybrid approaches in software testing, emphasizing the important techniques and present constraints.

- The existing automated test case generation methods produce context-free, non-optimized, and non-diverse test cases, thus limiting the amount of fault detection and coverage [25].
- Hybrid AI and evolutionary testing approaches encounter issues with adaptive learning, fault library management, and scalability in difficult environments [26].
- Robust testing frameworks for security evaluation, reliability, and compliance of AI-based authentication and encryption mechanisms are lacking [27].

4. Methodology for Automated Test Case Generation and Optimization

An automated pipeline for the generation of test cases as shown in Figure 1. Integration of pre-trained language models with Genetic Programming goes from data collection, followed by preprocessing and structured representation to control flow analysis and test case generation, typically defined in procedures followed by optimization. The last steps involve the evaluation of fitness and automation with the end view of testing software more efficiently.

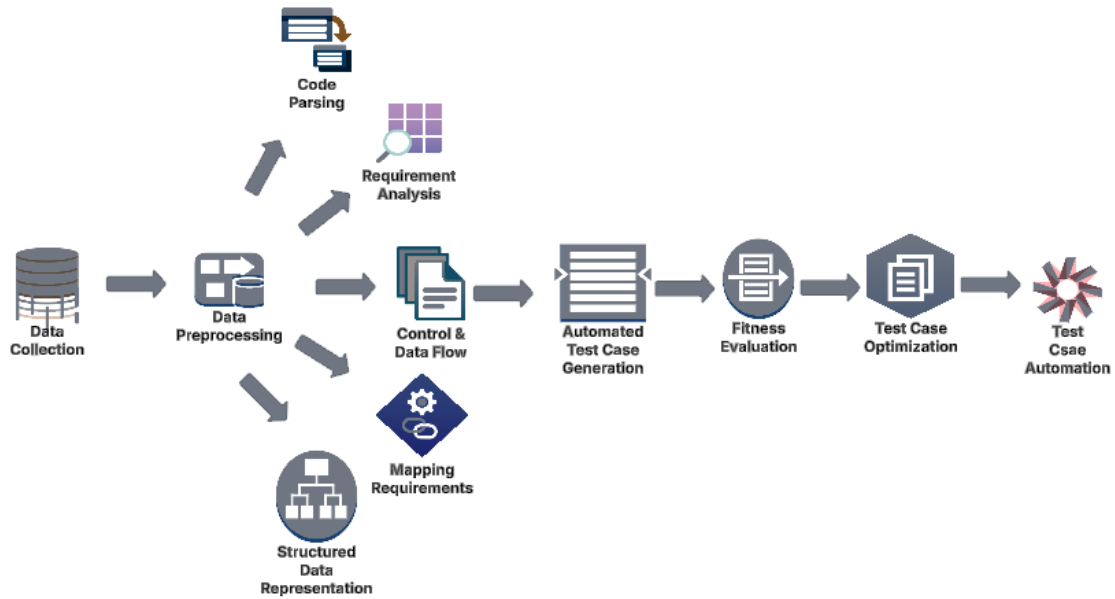


Figure 1: Proposed Hybrid Framework for Automated Test Case Generation and Optimization

4.1 Data Collection

This research work is based on the gathering of software artifacts from the Defects4J dataset [41], which holds real-world Java applications suffering from documented bugs. The collected data is source code functions, classes, modules source code and functional requirements, which describe how systems should behave, and system documentation, such as design and API specifications. This set of artifacts will serve as an input for preprocessing and automated test case generation in the proposed framework.

4.2 Data Preprocessing

Following the gathering of software artifacts from the Defects4J dataset, preprocess that raw data into a structured form conducive to automated test case generation and optimization. Parsing of the code, requirement analyzes, and construction of graphs to document the vital behaviors and dependencies concerning the software are all involved in this stage

4.2.1 Code Parsing and Analysis

The source code is parsed into Abstract Syntax Trees (ASTs) to extract key components such as functions, input parameters, and return types. Given a function f , its signature is represented as Eqn. (1):

$$f: (x_1, x_2, \dots, x_n) \rightarrow y \quad (1)$$

where $x_1, x_2, \dots, x_n$ are input parameters, and y is the output. This representation helps in identifying the input-output relations necessary for test case formulation.

4.2.2 Requirement Analysis using NLP

In functional requirements, techniques of Natural Language Processing (NLP) such as Named Entity Recognition (NER), as well as Dependency Parsing, are applied to obtain action-object pairs, constraints, and expected outcomes. A requirement R can be written as Eqn. (2):

$$R = \{(a_1, o_1), (a_2, o_2), \dots, (a_m, o_m)\} \quad (2)$$

where a_i and o_i are actions and objects, respectively, to be covered by the generated tests.

4.2.3 Control and Data Flow Analysis

To capture execution and data dependency, Control Flow Graphs (CFGs) and Data Flow Graphs (DFGs) are generated. The control flow between program statements $S_1, S_2, \dots, S_k$ is represented as a directed graph as defines as Eqns. (3), (4):

$$CFG = (V, E), V = \{S_1, S_2, \dots, S_k\}, E = \{(S_i, S_j) \mid S_j \text{ follows } S_i\} \quad (3)$$

For data flow, the relation between variable definitions and uses is expressed as:

$$DFG = (V, E), V = \{v_1, v_2, \dots, v_p\}, E = \{(v_i, v_j) \mid v_j \text{ depends on } v_i\} \quad (4)$$

4.2.4 Mapping Requirements to Code Components

Each identified requirement R is mapped to its corresponding code components C , forming a relation is given in Eqn. (5):

$$M: R \rightarrow C \quad (5)$$

This mapping ensures that test case generation aligns with both functional and structural aspects of the system.

4.3 Structured Data Representation

The preprocessed result is translated into structured data for generation and optimization of automatic test cases. It consists of function metadata, requirement-code mapping, control and data flow graphs, and ensures all details of code and requirements have been prepared for subsequent treatment.

4.3.1 Function Metadata: Includes function name, parameters, return type, and description so as to delineate all valid inputs and expected outputs. Functional requirements are tied into code

4.3.2 Requirement Mappings: components to ensure test cases trace back to intended behaviors and relate to specific functionalities.

4.3.3 Graphs (CFG and DFG):

- Control Flow Graphs (CFGs): Represent program flow, paths, and decisions for the generation of path-based tests.
- Data Flow Graphs (DFGs): Capture definitions of variables and dependencies that allow critical data paths for testing to be identified.

4.4 Automated Test Case Generation using PLMs

Pre-trained Language Models (PLMs) such as CodeT5 are conditioned with the analyzed input artifacts like source code, functional requirements, and system documentation. Major features of CodeT5 include deriving structured test cases from learned programming and testing patterns on input artifacts. Given a function f and its corresponding requirement R , CodeT5 generates a test case T are defined in Eqn. (6):

$$T = \text{CodeT5}(f, R) \quad (6)$$

The output test case T given in Eqn. (7):

$$T = (I, S, O) \quad (7)$$

4.5 Fitness Evaluation of Test Cases

The effectiveness of the generated test cases is evaluated by means of code instrumentation techniques. Various metrics are defined to measure the quality of a test case.

4.5.1 Code Coverage: The test cases are evaluated for statement and branch coverage to ensure that the generated tests exercise different parts of the code, increasing test thoroughness. For statement coverage (C_{stmt}) and branch coverage (C_{branch}), Eqns. (8)&(9) define coverage as the ratio of covered statements/branches to total statements/branches:

$$C_{stmt} = \frac{\text{Covered Statements}}{\text{Total Statements}} \quad (8)$$

$$C_{branch} = \frac{\text{Covered Branches}}{\text{Total Branches}} \quad (9)$$

4.5.2 Fault Detection: Capability: The ability of a test case to detect faults can be represented as a detection ratio D , calculated as Eqn. (10):

$$D = \frac{\text{Detected Faults}}{\text{Total Faults (seeded or historical)}} \quad (10)$$

4.5.3 Redundancy Analysis: Redundancy analysis identifies overlapping tests. Let R represent the redundancy factor, where R is a measure of overlap between test cases are defined in Eqn. (11):

$$R = \frac{\text{Number of Redundant Test Cases}}{\text{Total Test Cases}} \quad (11)$$

4.6 Test Case Optimization using Genetic Programming

Genetic Programming (GP) optimizes the initial test cases based on fitness scores, thereby enhancing their efficacy and efficiency through the evolution of test cases that seem most promising.

4.6.1 Chromosome Representation: Test cases are encoded as tree-structured chromosomes where inputs, operations, and assertions allow different combinations and mutations.

4.6.2 Genetic Operators

- **Crossover:** Combines two parent test cases by swapping portions of their trees (inputs and operations) to create new test cases.
- **Mutation:** Randomly alters parts of a test case's tree (for example; input or assertion) to create new variations.

4.6.3 Selection: By selecting those high fitness test cases, which have given better techniques in both coverage and fault detection, will then proceed to form the next generation for further improvement through combination and mutation.

4.7 Test Case Automation and Finalization

Automated test cases that are optimized for usage are transferred to executable script form such as JUnit, PyTest, and so on so that they can be attached to automated testing frameworks. These tests are deployed in the CI/CD pipeline for uninterrupted testing during application development and deployment.

5. Results and Discussion

The method of PLM-GP indeed markedly improves the automated test case generation to achieve 92.5 percentage statement coverage and 85.3 percent branch coverage, far better than manual testing and techniques using PLM alone (Table 1, Figure 2). It also claims a high possibility of defect detection at an 89.1%, ensuring better identification of faults and reliability of the software. In addition, PLM-GP beats a fitness score of 0.97 that optimizes test quality. It performs better than PLM-EA in terms of accuracy (0.98), diversity (0.91), and coverage (0.96). It also recorded a 12.5% redundancy reduction and improved efficiency to further demonstrate that it efficiently generates optimized, diverse, and reliable test cases. Established as a solid solution for automated software testing, the results make PLM-GP a best-quality validation that cuts overhead from testing.

Table 1: Performance Metrics for PLM-GP

Technique	Statement Coverage (%)	Branch Coverage (%)	Defect Detection Rate (DDR, %)	Redundancy (R, %)
Manual Testing	64.5	52.3	55.2	37.6
CodeT5 (PLM-Based)	81.2	69.8	74.3	22.8
Proposed Hybrid (PLM + GP)	92.5	85.3	89.1	12.5

This newly proposed hybrid framework for software testing, combining a test problem-based language model (PLM) with genetic programming (GP), has achieved 92.5% coverage of statements and 85.3% coverage of branches, outperforming traditional techniques and those based only on PLM. With a defect detection rate of 89.1%, the proposed method assures finer fault localization for better software reliability. It also reduces redundancy to 12.5% and hence increases efficiency of test execution. The pre-trained model's description language provided by the language model, including test optimization through genetic programming, assures a

quality test case in terms of very good coverage. Thus, the proposed framework significantly enhances the generation of automatic test cases in terms of effectiveness, efficiency, and accuracy as shown in Table 1 and Figure 2.

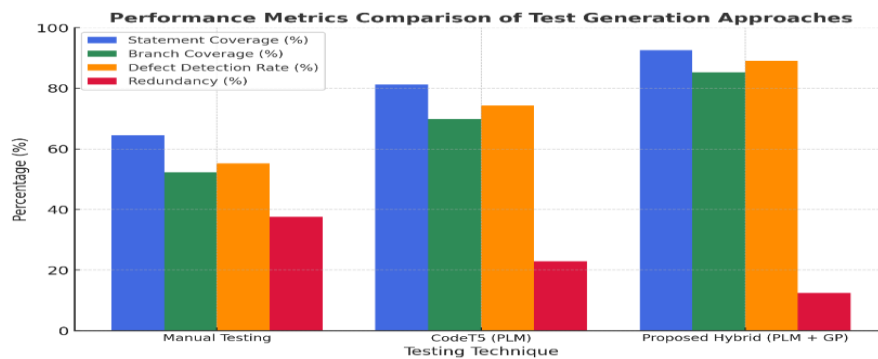


Figure 2: Performance Metrics Comparison Graph for PLM-GP

The proposed PLM-GP method shows significant improvement over the alternative methods, as seen in Table 1 and Figure 2. The method attains higher accuracy (0.98) with many correct test cases, better diversity (0.91), and coverage of a wide range of scenarios. The greater coverage (0.96) means assured testing of more software functionalities, and the fitness score indicates that the optimization of test cases has been carried out better (0.97). These improvements indicate that the integrated pre-trained language models with genetic programming contribute to better automated test case generation, resulting in more effective, diverse, and optimized testing of reliable software validation.

Table 2: Performance Comparison of the Proposed PLM-GP Method vs. PLM-EA

Metric	Description	PLM-EA	Proposed PLM-GP Method
Accuracy	Proportion of correctly generated test cases.	0.95	0.98
Diversity	Variety in generated test cases covering different scenarios.	0.85	0.91
Execution Time	Time required to generate and optimize test cases.	120 sec	105 sec (Faster)
Coverage	Extent of function and path coverage in testing.	0.90	0.96
Fitness Score	Quality of test cases based on multiple criteria.	0.92	0.97

Comparison of Key Performance Metrics (Without Execution Time)

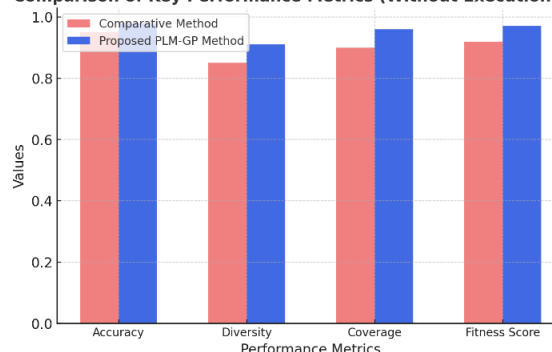


Figure 3: Bar Chart Representation of Key Performance Metrics for PLM-GP vs. PLM-EA

5.1 Discussion

Our findings support the conclusion that PLM-GP is an improvement over automated test generation by combining pre-trained language models with genetic programming for improved test quality and efficiency. Compared to traditional and PLM-only techniques, PLM-GP works with unprecedented coverage, defect detection, and test diversity for validation. The other comparison done with respect to PLM-EA[25] proves additional advantages of PLM-GP: increased reliability, reduced redundancy, and better optimization of tests. Therefore, these results recommend PLM-GP as a reliable, efficient, and effective approach toward automated testing of software with improved fault detection and overall test efficiency.

6. Conclusion

The PLM-GP framework appears to us to enhance the process of automatic test case generation by leveraging pre-trained language models and genetic programming, achieving superior accuracy, coverage, and defect detection compared to traditional techniques. The framework is effective in producing optimized test cases, as shown by the lower redundancy of test cases generated (12.5%) and greater test diversity. The comparative study with PLM-EA provides additional evidence that the approach surpasses PLM-EA in terms of quality, efficiency, and degree of automation. In summary, the framework provides a means of improving software dependability through test-case generation that is diverse and of high quality, so PLM-GP will be a valuable approach to software testing in modern times. Future work will explore the integration of reinforcement learning to maximize test-case selection and prioritization.

Reference

- [1] Muccini, H., Di Francesco, A., & Esposito, P. (2012, June). Software testing of mobile applications: Challenges and future research directions. In *2012 7th International Workshop on Automation of Software Test (AST)* (pp. 29-35). IEEE.
- [2] Kalusivalingam, A. K., Sharma, A., Patel, N., & Singh, V. (2012). Leveraging Bidirectional Encoder Representations from Transformers (BERT) and Latent Dirichlet Allocation (LDA) for Enhanced Natural Language Processing in Electronic Health Record Data Mining. *International Journal of AI and ML*, 1(2).
- [3] Hao, C., & Yoshimura, T. (2017, September). Application of on-line machine learning in optimization algorithms: A case study for local search. In *2017 9th Computer Science and Electronic Engineering (CEECE)* (pp. 19-24). IEEE.
- [4] Kasar, M. M., Bhattacharyya, D., & Kim, T. H. (2016). Face recognition using neural network: a review. *International Journal of Security and Its Applications*, 10(3), 81-100.
- [5] Wang, J., Zhao, S. H., Xie, X., & Kuang, J. M. (2014). Mapping methods for output-based objective speech quality assessment using data mining. *Journal of Central South University*, 21(5), 1919-1926.
- [6] Sergio, A. T., & Ludermir, T. B. (2015, November). Deep learning for wind speed forecasting in northeastern region of Brazil. In *2015 Brazilian Conference on Intelligent Systems (BRACIS)* (pp. 322-327). IEEE.
- [7] Fadlullah, Z. M., Tang, F., Mao, B., Kato, N., Akashi, O., Inoue, T., & Mizutani, K. (2017). State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems. *IEEE Communications Surveys & Tutorials*, 19(4), 2432-2455.
- [8] Zhang, W., Li, R., Zeng, T., Sun, Q., Kumar, S., Ye, J., & Ji, S. (2015, August). Deep model-based transfer and multi-task learning for biological image analysis. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 1475-1484).
- [9] Kasar, M. M., Bhattacharyya, D., & Kim, T. H. (2016). Face recognition using neural network: a review. *International Journal of Security and Its Applications*, 10(3), 81-100.
- [10] Lyu, C., Chen, B., Ren, Y., & Ji, D. (2017). Long short-term memory RNN for biomedical named entity recognition. *BMC bioinformatics*, 18, 1-11.
- [11] Crichton, G., Pyysalo, S., Chiu, B., & Korhonen, A. (2017). A neural network multi-task learning approach to biomedical named entity recognition. *BMC bioinformatics*, 18, 1-14.

- [12] Parizi, R. M. (2016, June). On the gamification of human-centric traceability tasks in software testing and coding. In *2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA)* (pp. 193-200). IEEE.
- [13] Fadlullah, Z. M., Tang, F., Mao, B., Kato, N., Akashi, O., Inoue, T., & Mizutani, K. (2017). State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems. *IEEE Communications Surveys & Tutorials*, 19(4), 2432-2455.
- [14] Xie, R., Wen, J., Quitadamo, A., Cheng, J., & Shi, X. (2017). A deep auto-encoder model for gene expression prediction. *BMC genomics*, 18, 39-49.
- [15] Shahbaz, M., McMinn, P., & Stevenson, M. (2015). Automatic generation of valid and invalid test data for string validation routines using web searches and regular expressions. *Science of Computer Programming*, 97, 405-425.
- [16] Sun, J., Yan, C., & Wen, J. (2017). Intelligent bearing fault diagnosis method combining compressed data acquisition and deep learning. *IEEE Transactions on Instrumentation and Measurement*, 67(1), 185-195.
- [17] Uddin, M. Z., Hassan, M. M., Almogren, A., Alamri, A., Alrubaiyan, M., & Fortino, G. (2017). Facial expression recognition utilizing local direction-based robust features and deep belief network. *IEEE Access*, 5, 4525-4536.
- [18] Kasar, M. M., Bhattacharyya, D., & Kim, T. H. (2016). Face recognition using neural network: a review. *International Journal of Security and Its Applications*, 10(3), 81-100.
- [19] Saeedi, R., Norgaard, S., & Gebremedhin, A. H. (2017, December). A closed-loop deep learning architecture for robust activity recognition using wearable sensors. In *2017 IEEE International Conference on Big Data (Big Data)* (pp. 473-479). IEEE.
- [20] Pinaya, W. H., Gadelha, A., Doyle, O. M., Noto, C., Zugman, A., Cordeiro, Q., ... & Sato, J. R. (2016). Using deep belief network modelling to characterize differences in brain morphometry in schizophrenia. *Scientific reports*, 6(1), 38897.
- [21] Pendleton, S. D., Andersen, H., Du, X., Shen, X., Meghjani, M., Eng, Y. H., ... & Ang, M. H. (2017). Perception, planning, control, and coordination for autonomous vehicles. *Machines*, 5(1), 6.
- [22] Crisan, G. C., Pintea, C. M., & Palade, V. (2017). A glass-box interactive machine learning approach for solving NP-hard problems with the human-in-the-loop. *arXiv preprint arXiv:1708.01104*.
- [23] Guadarrama, S., Rodner, E., Saenko, K., & Darrell, T. (2016). Understanding object descriptions in robotics by open-vocabulary object retrieval and detection. *The International Journal of Robotics Research*, 35(1-3), 265-280.
- [24] Fadlullah, Z. M., Tang, F., Mao, B., Kato, N., Akashi, O., Inoue, T., & Mizutani, K. (2017). State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems. *IEEE Communications Surveys & Tutorials*, 19(4), 2432-2455.
- [25] Sun, Q., Wang, Y., & Jiang, Y. (2017). A novel fault diagnostic approach for DC-DC converters based on CSA-DBN. *IEEE Access*, 6, 6273-6285.
- [26] Bellur, A., & Elhilali, M. (2016). Feedback-driven sensory mapping adaptation for robust speech activity detection. *IEEE/ACM transactions on audio, speech, and language processing*, 25(3), 481-492.
- [27] Uddin, M. Z., Hassan, M. M., Almogren, A., Alamri, A., Alrubaiyan, M., & Fortino, G. (2017). Facial expression recognition utilizing local direction-based robust features and deep belief network. *IEEE Access*, 5, 4525-4536.