Journal of Information Technology Management

Print ISSN: 2008-5893

Online ISSN: 2423-5059

In-Depth Analysis of Various Artificial Intelligence Techniques in Software Engineering: An Experimental Study

Mohd Mustaqeem *

*Corresponding Author, Ph.D. Scholar, Department of Computer Science, Science, Aligarh Muslim University (AMU), Aligarh, U.P, India. E-mail: mohdmustaqeem34@gmail.com

Tamanna Siddiqui

Professor, Department of Computer Science, Aligarh Muslim University (AMU), Aligarh, U.P, India. E-mail: tsiddiqui.cs@amu.ac.in

Najeeb Ahmad Khan

Associate Professor, Faculty of Engineering & Technology at Arunachal University of Studies, Namsai, Arunachal Pradesh, India. E-mail: naahkh@gmail.com

Deepak Kumar

Professor, Amity University Uttar Pradesh, Noida, India. E-mail: deepakgupta_du@rediffmail.com

Abstract

In this paper, we have extended our literature survey with experimental implementation. Analyzing numerous Artificial Intelligence (AI) techniques in Software Engineering (SE) can help understand the field better; the outcomes will be more effective when used with it. Our manuscript shows various AI-based algorithms that include Machine Learning techniques (ML), Artificial Neural Networks (ANN), Deep Neural Networks (DNN) and Convolutional Neural Networks (CNN), Natural Language Processing (NLP), Genetic Algorithms (GA) applications. Software testing using the Ant Colony Optimization (ACO) approach, predicting software maintainability with Group Method of Data Handling (GMDH), Probabilistic Neural Network (PNN), and Software production with time series analysis technique. Furthermore, data is the fuel for AI-based model testing and validation techniques. We have also used the NASA dataset promise repository in our script. There are various applications of AI in SE, and we have experimentally demonstrated one among them, i.e., software defect prediction using AI-based techniques. Moreover, the expected future trends have also been mentioned; these are some significant contributions to the research.

Keywords: Software Engineering, Defects Prediction, Artificial Intelligence, ML, ANN, DNN, CNN

Journal of Information Technology Management, 2023, Vol. 15, Issue 3, pp. 162-181
Published by the University of Tehran, Faculty of Management
doi: https://doi.org/10.22059/jitm.2023.93632
Article Type: Research Paper

Received: April 03, 2023
Received in revised form: June 13, 2023
Accepted: July 20, 2023
Published online: August 26, 2023

Introduction

In today's world, every person is surrounded by modern-day technologies. Social media, e-commerce purchasing, e-health, and online transactions are the daily life parts of people. All these things require some online platform information system software so that people can take advantage of this modern-day thing. But as technology is emerging daily, data is generated continuously, and hackers and crackers are constantly trying to compromise the security of these systems. So, there is a need to prevent data and protect software from malfunctioning. To prevent software from any failure, testing is one of the methods. According to Kuhn et al. (2004), exhaustive testing of software is done if an error occurs due to some or all parameters testing team tests those parameters rather than complete software testing. Finding those parameters and combinations is also hectic and time-consuming, and the chance of success is not very high. To maintain quality and save cost, time, and labor, exhaustive testing of software is not a good idea. It is not feasible, and testing must be automated. To overcome this issue, an AI-based technique ant colony optimization (ACO) (Srivastava & Baby, 2010) approach is used; this approach creates some test sequences due to which complete software is covered for testing. The development of better software and its success mainly depends on knowledge, practice, experience, and learning of systems which results in providing true information in accurate time, at the correct place, and according to the needs of the recipient. Sometimes this approach remains less effective; there is another approach of SE decision support (SEDS) (Li et al., 2017); it combines various aspects of software systems, like models. Experimental techniques, with intelligent methods of analysis and interpretation to know the results of decisions that help reduce the chances of future mistakes. After making a decision, Software coding is one of the essential tasks that require a programmer or a team of a programmer with proper knowledge of different types of programming languages who can analyze, write thousands of lines of code, and change accordingly, which require lots of time, energy having greater chances of error.

On the other side, using an AI-based automated intelligent expert system for computer programmer assistants can resolve almost all the above issues. A small defect in the software may lead to software collapse; the motive of software development is destroying when the software faces any defect; the software industry and other software product owners cannot

bear the loss of time, money, and labor. When the company runs on a software platform during working hours, a small defect in software may collapse the entire company. So, the prediction of software defects, in the beginning, is one of the significant challenges in SE; many software models design uses previous data or history from the database, mine it, and detect the defects in software (Kim et al., 2011) in this traditional defect prediction technique, there is a chance of not getting complete success, it may be time taking process chances of failure, to develop the high-quality product which is defect-free, feasible to the stakeholders and the results may be good when we use the intelligent system: there are various AI-based techniques to predict the defect in the software, in the manuscript (Siddiqui et al., 2021) and (Bibi et al., 2006) we have reviewed machine learning Classification algorithm which identity, determines, evaluate advance and manage the system. Planning for a software project is one of the essential activities of SE. Poor planning may lead to project collapse; estimating the cost of the system is one of them. It is the process of predicting the necessary effort to develop a software system. If the cost and effort are not estimated correctly with time, it may cause project failure.

There are many software cost estimation models developed. If a model is selected must have accuracy in estimation, and the cost of the product does not exceed the limit. The COCOMO model is one of them, but it has some limitations. It cannot deal effectively with incomplete, indefinite data. The COCOMO model must be standardized to estimate it precisely (Gharehchopogha & Khalifehlou, 2012). To overcome this issue, an intelligent method is used for early prediction of cost estimation of any software. We have used regression analysis techniques in our paper. The regression analysis model takes the dataset from the NASA project and is classified as the trained and testing data (Strnad & Guid, 2010).

The rest of the paper is organized as follows: Section 2 represents the related work. Section 3 presents the applications of AI techniques in SE. Section 4 presents the Impact analysis. Section 5 describes the datasets, section 6 presents the experimental study, section 7 presents the Results and Discussion, section 8 presents the future trends, and section 9 concludes.

Literature Review

Nowadays, SE and AI are used together to get better and faster results. According to N. Guide (Bennett & Rajlich, 2000), by using AI, we can easily predict the best team among many; in cricket and business, AI-based techniques are used to predict the best team. The success of any software and another project mainly depends on planning and team; choosing a solid team is essential and can complete the project on time, more efficiently, and accurately. It is difficult for a manager and team selector to select a perfect team traditionally from the cluster; we have reviewed the fuzzy-genetic analytical model technique for software team building. It works on the team members' historical and modern personal attributes, which can predict the best team. Once a team is allotted to develop software, maintainability of software is also a

significant phase of any software. It depends entirely on its maintenance; if the software is not well maintained, it will not be used in the future; for developers and owners, this is one of the significant issues; to some extent, this issue is resolved in SE with some methods. However, if we talk about AI, we have used the (Malhotra1 & Chug2, 2012) Group Method of Data Handling (GMDH), Genetic Algorithms (GA), and Probabilistic Neural Network (PNN) approach to maintain the software. In this technological world, due to the enormous usage of software systems, companies are developing a considerable amount of software, so software forecasting and its trends and quality must be known to companies which can be predicted from different forecasting techniques.

Nevertheless, we have reviewed time-series analysis with quality forecasting in open-source software (Parizi & Ghani, 2010)to forecast software trends and quality in the market for software production. We have reviewed the above application of AI-based techniques in SE and gathered information on a single platform which will be helpful for further research. As we know, the basic definition of SE is the principles used in developing software, complex and human-oriented activity; using artificial intelligence, many software development activities can be improved (Saini, 2016). The hybrid techniques are also used for predicting software defects using AI techniques (Mustaqeem & Saqib, 2021). When computational work is performed by humans and requires intelligence, the artificial intelligence field comes into consideration. The main focus of AI is creating an intelligent machine, a computer system that can understand natural language like a human. Many sub-fields of AI have applications in SE, like machine learning; it deals with the issues of how to develop systems that can solve problems through self-learning and previous experience. It is effectively applied in numerous parts of SE, from pattern recognition behavior extraction, testing of software, and prediction of defects. Machine learning algorithms are used for innumerable data mining problems, where many databases may contain valuable information that can be recognized automatically.

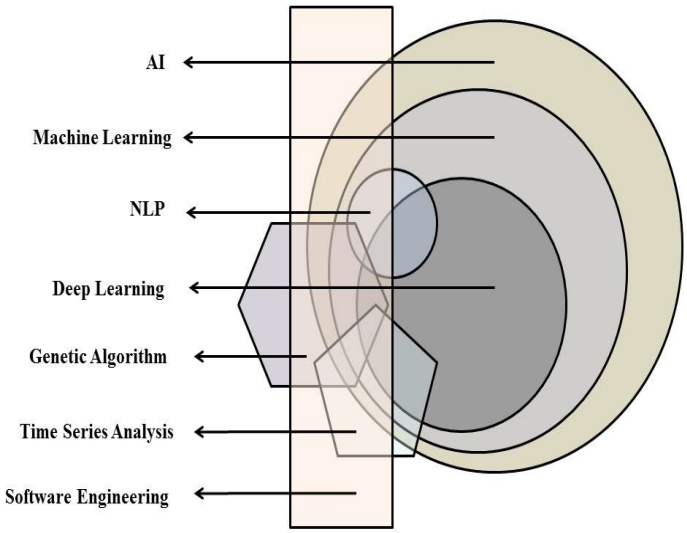
ML algorithms explore areas not explored by humans and do not have the idea to develop algorithms in areas where the programs must dynamically adapt to changing conditions. The algorithms which are used in machine learning are Concept Learning (CL), Decision Trees (DT), Artificial Neural Networks (ANN), Bayesian Belief Networks (BBN), Reinforcement Learning (RL), Genetic Algorithms (GA), and Genetic Programming (GP), Instance-Based Learning (IBL), Inductive Logic Programming (ILP), and Analytical Learning (AL) (Nassif et al., 2012).

Application of Artificial Intelligence techniques in SE

In the past few years, SE has created a different image in the market; its rising rate is high. It starts from software development to modification data generated learning problems which may result in learning algorithms, as shown in Figure 1.

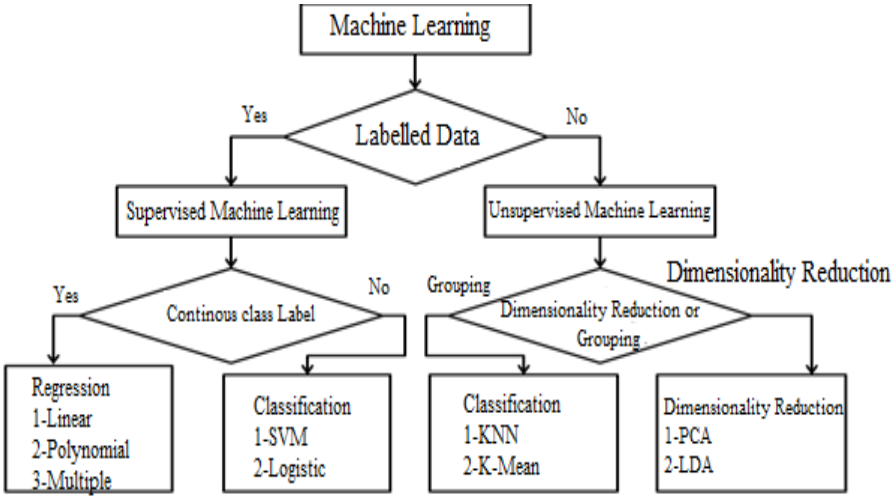
- Machine Learning
- Deep Learning
- Natural Language Processing
- Genetic Algorithms
- Time Series Analysis

Figure 1.
AI techniques in SE



Machine Learning: Machine learning algorithms have been essential in improving SE. Earlier, old approaches were used to find software defects, and software cost estimation was very time-consuming and not efficient. However, now we can apply regression analysis, classification, clustering, and dimensionality reduction techniques to enhance the capabilities, as shown in Figure 2.

Figure 2.
Hierarchy of Machine Learning Algorithms



Deep Learning: Deep learning is a subset of machine learning based on Artificial Neural Networks (ANN) and representation learning. ANN is used in biological systems to process information and send messages to different nodes. There are a lot of differences between the brains of living things and ANNs. Neural networks are fixed and symbolic, while the brains of living things are analog and change over time. Deep learning uses multiple layers in the network; deep learning algorithms use unknown datasets to find helpful information at numerous layers; deep learning is feature learning; it can automatically pull features from raw data. Deep learning algorithms like ANN, DNN, CNN, and GCNN are used in SE. These algorithms help SE keep up with modern technologies and make SE a more realistic, robust, dynamic, and automated field.

ANN: According to the paper, software efforts can be predicted using ANN from use case diagrams depending on the Use Case Point (UCP) model. The software size, how well it works, and how complicated it is can be used as inputs, and the software efforts can be used to predict the results. Multiple linear regression models with three independent variables and one dependent variable were used to measure ANN. ANN is experienced against the regression model and Use Case Point (UCP) model based on MMER and PRED, and results show that the ANN model performs 8% and 50% against the regression model and UCP (Nassif et al., 2012). ANN or Multi-layer perceptron is also used for classification purposes and can be used for software defect prediction; figure 3 shows the ANN input layers with corresponding weights and output. Mathematically, we can represent it as:

$$\sum = (y_i \times w_i) + (y_2 \times w_2) + \cdots + (y_n \times w_n) \quad (1)$$

Where y_i are the inputs and w_i are the allocated weights. The values of the weights play a crucial role in output.Row vectors for input are represented as $y = [y_i, y_2, \cdots, y_n]$ and $w = [w_i, w_2, \cdots, w_n]$, their dot product can be represented as:

$$y.w = (y_i \times w_i) + (y_2 \times w_2) + \cdots + (y_n \times w_n) \quad (2)$$

The final dot product is represented as:

$$\sum = y.w \quad (3)$$

By implementing the bias c with the dot product's summation, we get p, and the equation can further be written as:

$$p = y.w + c \quad (4)$$

The value of p can be passed to the activation function; we will implement the sigmoid activation function here:

$$\bar{x} = \sigma(p) = \frac{1}{1+ e^{-p}} \quad (5)$$

σ denotes the activation function and predicted value ($\overline{x}$) the output value after forward propagation.

A loss function can be used for deviation from the actual result that we can show using mean square error:

$$MSE_i = (x_i - \bar{x_i})^2 \quad (6)$$

For the entire training dataset, the loss function's average, known as cost function R, can be calculated:

$$\text{R=MSE}=\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x_i})^2 \quad (7)$$

The gradient of R wrt weights can be calculated using partial derivation.

$$\frac{\partial R}{\partial w_i} = \frac{\partial R}{\partial \hat{x}} \times \frac{\partial \hat{x}}{\partial p} \times \frac{\partial p}{\partial w_i} \quad (8)$$

$$\frac{\partial R}{\partial \hat{x}} = \frac{\partial R}{\partial \hat{x}} \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x_i})^2 = 2 \times \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x_i}) \quad (9)$$

$$\frac{\partial \hat{x}}{\partial p} = \frac{\partial\, \sigma(p)}{\partial p} \quad (10)$$

$$\frac{\partial p}{\partial w_i} = \frac{\partial (p)}{\partial w_i} \quad (11)$$

For optimization, we can select the hyperparameter, which is also known as the learning rate (β).

Weights and bias can be updated by backpropagation, and gradient descent is used until convergence:

$$w_i = w_i - \left(\beta \times \frac{\partial R}{\partial w_i}\right) \tag{12}$$

$$\text{c=c} - \left(\beta \times \frac{\partial R}{\partial c}\right) \tag{13}$$

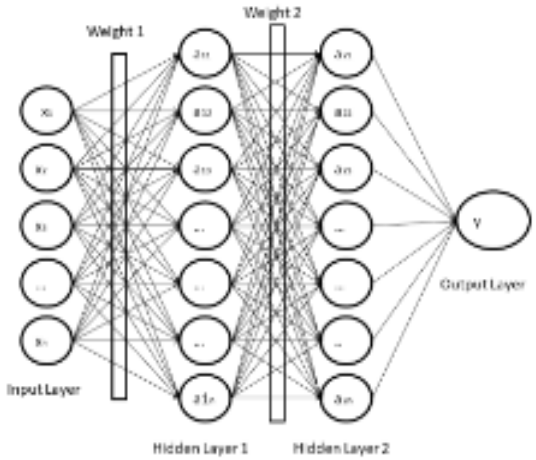


Figure 3. ANN with Input, Weight and Output layers

DNN: The DNN, an ANN, is another deep learning algorithm used in automotive software. It makes nonlinear compound patterns by hiding many units between the input and output layers. Using DNN, we can change the hidden layers, units per layer, and connections per unit. Its organization is flexible in these ways, as shown in (Falcini et al., 2017).

Mathematically, we can say that DNN is a computational function. And it has three layers input layer, a hidden layer, and an output layer.

Let $M^k(y): P^{ain} \rightarrow P^{aout}$ beM-layer NN, with M_k neurons in the kth layer ($M_0 = a_{in}, M_k = a_{out}$). Weight matrix and vector bias can also be denoted in the kth layer by $W^k \in P^{M_k \times M_{k-1}}$ and $C^k \in P^{M_k}$. By applying the nonlinear activation function, we can get the following values:

Input/Initial layer:

$$M^0(y) = y \in P^{ain}, \tag{14}$$

Hidden/Middle layer:

$$M^k(y) = \sigma(W^k M^{k-1}(y) + C^k) \in P^{M_k} \text{ for } 1 \leq k \leq K-1 \tag{15}$$

Output layer:

$$M^L(y) = (W^K M^{K-1}(y) + C^K) \in P^{aout} \tag{16}$$

CNN: Good quality software is based on software requirements, including information, effects, and expectations about software development. Large or small, the software requires lots of human effort and involvement. We looked at the CNN model, which uses the PROMISE dataset well on SRC (Navarro-Almanza et al., 2017), to automate and classify software requirements without much human or less human involvement. The mathematical representation of CNN is by using the * sign. If we have Y as an input image and a filter t, then the expression would be like this:

$$M = Y * t \tag{17}$$

The linear transformation can be done:

$$M = W^T.Y + c \tag{18}$$

The sigmoid activation function can be applied as follows:

$$\text{f(x)}=\frac{1}{(1+e^{-y})} \tag{19}$$

$$\text{M1}= W^T.f(x) + c \tag{20}$$

$$\frac{\partial M1}{\partial f(x)} = W^T \tag{21}$$

$$\text{Output=Sigmoid(M1)} \tag{22}$$

Natural Language Processing (NLP): For designing, developing, and sequentially testing the software product, industries use a process called software development life cycle (SDLC), which is the most essential, stepwise chronological structure in SE (Sadiku et al., 2018). In SDLC, documentation and plain text are the final articles that would be helpful to apply NLP in every phase of SDLC. In our paper, we have reviewed NLP applications in SDLC (Pressman & Roger S., 2010), which use textual characters as input for NLP, and the artefacts show the textual document generation. There are various phases of textual artefacts'; we are reviewing some below. Text processing can be done to transform different words into one speech form. We can also measure the similarities and differences among the strings.

Text similarity measuring can be done using cosine similarity metrics.

$$\cos(\theta) = \frac{\overline{P.Q}}{|P||Q|} \tag{23}$$

We can convert words into numerical vectors using vectorization methods like "bag of words" and "TF-IDF." To find feature independence in text classes, we have used Naïve Bayes

$$\mathrm{P(a|b)} = \frac{P(a|b)\times P(a)}{P(b)} \tag{24}$$

$$\arg\max\,[P(R_l \times \Pi P(y_i|R_l))] \tag{25}$$

Table 1.
Analysis Phase Textual Artifacts

Document/ Artifact	Author
Requirement Document	System Analyst
Software Requirement Specification	System analysts Business manager
Use Case Description	System Analyst
Acceptance Test Cases	Tester

Table 2.
Design Phase Textual Artifacts

Document/ Artifact	Author
Software Design Specification	Designer
UML Diagrams	Design Engineer
Design-level Test cases	Tester

Textual artifacts and a cost requirement analysis are part of the umbrella activities. As discussed in SDLC (Pressman & Roger S., 2010), NLP uses a textual format generated by the above activities and can be automated using tools and techniques. Using a machine translation, it can be turned into another natural language, according to Yalla and Sharma (2015).

Genetic Algorithms: Genetic Algorithms are part of soft computing instead of AI. However, this survey can still be considered because AI algorithms solve some problems with SE optimization. Biological theories inspire genetic algorithms-Darwin's principle of best fittest for survival, and many times, it is used to find out the best-fitted population before applying any AI-based techniques, as in the following examples: Mathematically, we can show it as:

$$(a_1, a_2, \cdots, a_n)_2 = \left(\sum_{j=0}^{M} a_i 2^j\right) = y' \tag{26}$$

In the above equation ($a_j \in \{1,0\}$) is chromosome gene,

We can convert the y' value in the interval (0;1) from the following equation:

$$y = k_{low} + y' \frac{K_{upper}}{2^M - 1} \quad (27)$$

$k_{low} = 0 \; and \; k_{upper} = 1$

Now, the combination will become o's, and 1'sother combinations will also be in the range: (k_{low}, k_{upper}).

Software Testing using Ant Colony Optimization (ACO): In SE, many human activities are error-prone, and software engineers do many tasks. Like these activities, they should focus on whether the valuable data collected helps decision-making. When the software testing is done, heterogeneous data is collected. It keeps track of where and when the error happens. This information can be used to find interesting patterns and test information for more significant testing problems. As L. C. Briand talks (Jalote, P., 2012), an automated decision-making algorithm is used to find errors, faults, and possible ways to improve test specifications and decide the order of importance for test cases.

According to H. Li and C. P. Lam (Briand, 2008), software testing involves making test data, running tests with the test data and the software being tested, and evaluating the results of the tests. When testing software, the main goal is to choose test cases that will find the most module bugs. For example, it is possible to make an exciting set of test cases, but it is expensive and takes a long time, so cost optimization automation is needed (Li & Lam, 2014). We have reviewed a paper (Briand, 2008) in which H. Li and C. P. Lam discuss ACO; its performance is replicated with the real Ant for test case generation. Various activities consider the ACO algorithm, such as converting testing cases into a graph; a heuristic measure of the path through the chart; a mechanism for creating possible solutions efficiently; by this approach, the test pattern can be generated for state-based software testing. Mathematically, we can demonstrate the algorithm using:

$$P_{k,l} = \frac{(\tau_{k,l}^{\gamma})(\eta_{k,l}^{\delta})}{\sum(\tau_{k,l}^{\gamma})(\eta_{k,l}^{\delta})} \quad (28)$$

$It \; is \; the \; (\tau_{k,l}^{\gamma})$ pheromone amount on edge k, l

γ it is used to control the influence of the parameter of $(\tau_{k,l}^{\gamma})$.

The desirability of the edge (k, l) is defined by $(\eta_{k,l}^{\delta})$:

δ control the influence of $(\eta_{k,l}^{\delta})$

The pheromone updating is done using the following:

$\tau_{k,l}$= (1-ρ) $\tau_{k,l}$ + $\triangle_{k,l}$ (29)

$It\ is\ the\ (\tau_{k,l}^{\gamma})$ pheromone amount on edge k, l

ρ denotes the rate of evaporation of pheromone.

$\triangle_{k,l}\ pheromone\ amount\ deposit$

$$\tau_{k,l} = (1-\varphi).\tau_{kl} + \varphi.\tau_0 \quad (30)$$

$\varphi \in (0,1]$, pheromone decay coefficientτ_0 initial value.

Software maintainability using Group Method of Data Handling (GMDH), Genetic Algorithms (GA), and Probabilistic Neural Network (PNN):

Another subfield, like deep learning, also has applications in SE. We have reviewed many papers on software maintainability using different algorithms. When software works well and is easy to change, its performance stays good throughout its working life and meets the customer's needs. When this happens, we call the software "maintained." The enormous software maintenance cost is much more than the software developed. Observing the software metrics during the development phase is essential to controlling maintenance costs. This is done in many industries; they use tools and techniques to predict software maintainability during software metrics design (Horgan et al., 1994; Briand et al., 2001). We looked at the papers on the Group Method of Data Handling (GMDH), the Genetic Algorithm (GA), and the Probabilistic Neural Network (PNN) to keep the link between software metrics and maintainability. In GMDH, a polynomial model function has been developed, which can bring. The predicted value of the output is close to the actual value of the production. It applies to the link and genetic module in the polynomial term to decide layers. GA optimization is based on Darwin's evolution theory. Natural selection and genetics are the basis for searching for the most acceptable solution and a set of software metrics that gives the best arrangement. It first found the location of software metrics solutions for object-oriented software maintainability (Malhotra1 & Chug2, 2012). The latest algorithm, based on a neural network, is PNN; it works on the same principle as the human brain intelligence for new problem-solving. Neurons can also learn from problems that have already been solved and use that knowledge to develop new ones.

$$\delta_1 = \frac{\partial}{\partial w_k^l}\frac{1}{2}||h_{y,a}(p) - q||^2 = -(q_k - \alpha_k^l)g'(w_k^l) \quad (31)$$

$$\delta^m = ((n^l)^T\delta^{k+1})\ g'(w^m) \quad (32)$$

Function g' is found by GMDH, does not use actual function (g); instead, it uses approximation for output (o).It contains input vector V=$(v^1, v^2, v^3, \cdots, v^n)$, and it is close to the actual output, Q(TOC, R1, R2), for multi-input data:

$$o_i = g(v^{i1}, v^{i2}, v^{i3}, \cdots, v^{in}) \text{ (i =1,2,3,}\cdots, N) \quad (33)$$

For any input vector V=($v^{i1}, v^{i2}, v^{i3}, \cdots, v^{in}$), GMDH can be trained as:

$$o_i = g'(v^{i1}, v^{i2}, v^{i3}, \cdots, v^{in}) \text{ (i =1,2,3,}\cdots, N) \quad (34)$$

Determination of GMDH, which can minimize the square difference.

$$\sum_{i=0}^{N}\left[g'\left(v^{i1}, v^{i2}, v^{i3}, \cdots, v^{in}\right) - o_i\right]^2 \rightarrow min \quad (35)$$

The equation (30) can be resolved as:

$$\text{o = Y } (v^i, v^j) = b^0 + b^1 v^i + b^2 v^j + b^3 {v^i}^2 + \cdots \quad (36)$$

3.5 Time Series Analysis: With time, software demand is increasing drastically; almost every sector, like business, education, and the government, is working on software to overcome the direction and improve software productivity and excellence; data mining algorithms are used in SE tasks. Data should be in an expressive form, which can be done using the pattern-mining domain and enhanced capability of mining algorithms. Practical data mining algorithms are required to aid in analyzing the massive amounts of SE. Data to generate the best pattern, mine sequences, graphs, and text. Also, data-mining techniques include time-series data analysis, which keeps track of specific points in time. As we all know, data is made every second. From this time, series data mining can generate essential information for future use. Software trends can be analyzed and developed according to the recipient's needs. Data plotted against time can generate an excellent approach to testing the movement. Using time-series data analysis can find the irregularity in the data and forecast the pattern for future development software. Open-Source Software (OSS) uses statistical time series analysis to automate modeling and analysis and make predictions about software quality assurance. How software is tested now makes this more complicated; in the future, time-series analysis can predict software quality early and improve software productivity. It can reduce human efforts and involvement by predicting software excellence early.

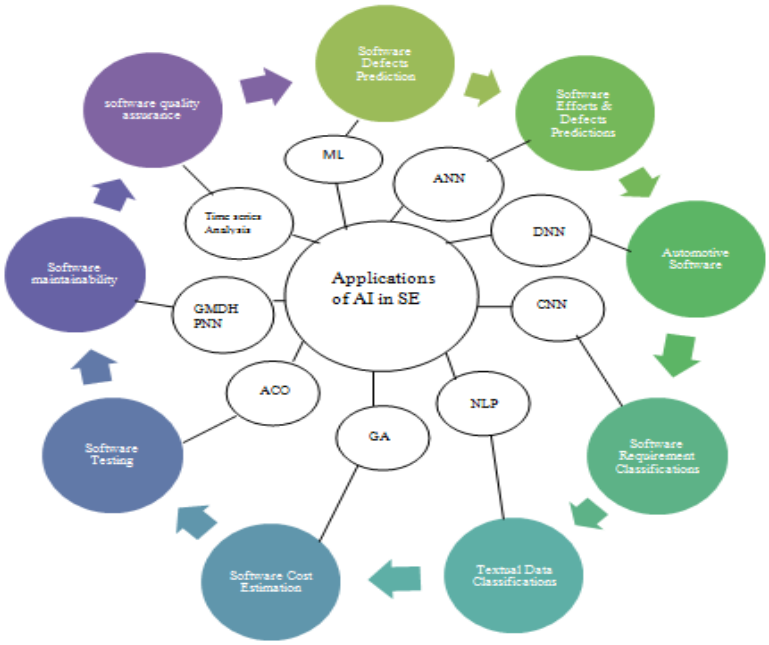
Impact Analysis of Various AI Techniques SE

We have used various AI techniques to correlate with SE applications in the given manuscript. In the given Table 3 shows the analysis.

Table 3.
AI techniques on SE

AI Techniques	Software Engineering
ML	Software defect prediction and software cost prediction can be made using ML techniques
ANN	Software Efforts Predictions using use case diagrams and software defects prediction
DNN	Automotive Software
CNN	Software Requirement Classifications
NLP	Textual Data Classifications
Genetic Algorithms	Software Cost Estimation
Ant Colony Optimization	Software Testing
Group Method of Data Handling (GMDH), Genetic Algorithms (GA), and Probabilistic Neural Network (PNN)	Software maintainability
Time Series Analysis	Automate the Modeling, analyze forecast software quality assurance of Open-Source software

Figure 4.
AI tools in SE



From the above discussions, Table 3 and Figure 4, we can say that AI and its sub-techniques can be used as a booster, effective, efficient, and time-saving SE methods.

Dataset

The working of the AI-based model purely depends on the datasets. Datasets are the fuel to test the performance of the models. There are so many types of models that have been developed to solve various problems. In the manuscript, we talked about software defect prediction datasets and how they were tested on a model that had already been made. We used our study's PROMISE software defect dataset repository (Horgan et al., 1994). The KC1, CM1, JM1, and PC3 datasets represent what we extract data, divided into training and testing

datasets used for SDP. Moreover, the datasets we used for our computation present the following features in Table 4.

Table 4.
PROMISE SDP features details

Feature Name	Description
LOC	Module total number of line count
Iv(g)	(McCabe) complexity design analysis
Ev(g)	McCabe complexity
N	Module numeral operators
v(g)	cyclomatic complexity measurement (McCabe)
D	Difficulty Measurement
B	Effort's Estimation
L	Length of Program
V	Volume
I	Intelligence Measurement
E	Effort measurement
Locomment	Software module line of comment
Loblank	number of total blank lines in the module
uniq_op	number of total unique operators
uniq_opnd	number of total unique operand
T	Estimator of Time
Branchcount	number of total branch in the software module
total_op	number of total operators
Total_opnd	number of total operators
Locodeandcomment	number of total line of code and comments
Defects/Problems	defect regarding information, whether it is present or not

The dataset description may include information such as dataset name, number of modules, and number of defective and non-defective classes, with their percentage shown in Table 5.

Table 5.
PROMISE Software Defect Prediction Dataset Details

Dataset Name	Total values	Non-defective	Defective	%Non-defective	%Defective
CM1	1988	1942	46	97.6	2.4
PC3	1077	943	134	87.5	12.5
KC1	2109	1750	359	82.9	17.0

Experimental Demonstration

Performance Metrics: In this section, we have demonstrated the performance metrics used in our experiment. The confusion matrix is used to measure the common five evaluation metrics.

Table 6.
Confusion Matrix

Actual Value	Predicted Value	
	Non-Defective	Defective
Non-Defective	False Negative (FN)	True Positive (TP)
Defective	True Negative (TN)	False Positive (FP)

Accuracy: The number of correct answers given in a classification. In a confusion matrix, there are two types of solutions. True Positive (TP) (where the defective value has been identified as defective) and True Negative (TN) (where the defective value has not been determined as defective) (where the non-defective value is specified as non-defective). The following formula can be used to calculate accuracy:

Accuracy=((TP)+(TN))/((TP+FN+FP+TN))

Precision: Measure correctly answered defective values to the total predicted defective values in the given classification. This can be represented as:

Precision=((TP))/((TP)+(FP))

Recall: Measure truly answered defective values, all actual defective values in the classification. This can be represented as:

Recall=((TP))/((TP)+(FN))

F1-score: Weighted average Precision and Recall. It can be represented as:

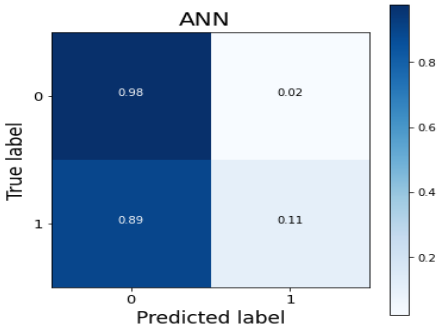
F=(2*Precision*Recall)/ (Precision + Recall)

Implementation

We have implemented the given datasets using Python programming and the Spyder platform on our AI-based model shown below.

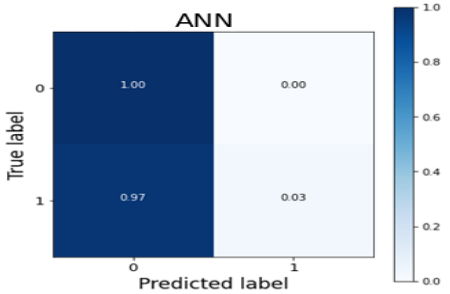
Dataset CM1: The CM1 dataset is used in the proposed AI-based model. Figure 5 shows the confusion matrix.

Figure 5.
CM1 Confusion Matrix using AI-based Model



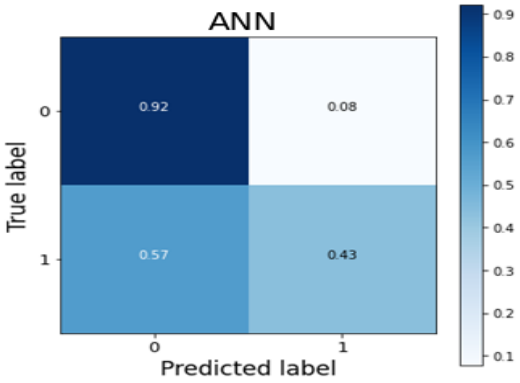
Dataset PC3: The PC3 dataset is used in the proposed AI-based model. Figure 6 shows the confusion matrix.

Figure 6.
PC3 Confusion Matrix Using AI-Based Model



Dataset KC1: The KC1 dataset is used in the proposed AI-based model. Figure 7 shows the confusion matrix.

Figure 7.
KC1 Confusion Matrix Using AI-Based Model



Results and Discussion

The experimental study of NASA repository datasets CM1, KC1, and PC3 on the proposed AI-based model shows the following results in Table 7.

Table 7.
Evaluation measures using considered metrics with an AI-based model

	Performance of AI-based Model			
	Evaluation Metrics			
Datasets	Precision	Recall	F1-score	Accuracy
CM1	88	98	93	87
KC1	91	92	91	85
PC3	89	88	94	88

From the above experimental study, we can make the discussions that AI techniques can effectively implement in software engineering tasks that can enhance performance and give promising results.

Future trends

We have seen the many uses of AI-based algorithms in the SE field. But there are a few doubts in our minds about whether AI will be helpful for SE in the future or not. Will there be any trend of AI that will be used in SE? From our knowledge, there may be some possible trends in the future, which we have discussed below.

Automated error-correcting and intelligent software: In the future, companies that make software will make intelligent AI-based software that can automatically fix mistakes, works quickly, and is efficient and reliable. Furthermore, governments may use AI-based software in their systems, such as government offices, educational institutions, national security, agriculture, and the environment, to combat cybercrime and space-related activities.

AI-based intelligent software will be timely-efficient, working efficacy will be enhanced, its low maintenance cost and fewer people may run them, and more occasional efforts will be required from these qualities. It may be possible that the country's economy will increase by using innovative education with AI-based software in the future, which will enhance the understanding and intelligence level of the learner concept of intelligent farming, like autonomous tractors, automatic watering, robotic harvesters, and seeding robots. e.g., Agrobot is the first robot used for successfully harvesting strawberries and many more; it will fulfill the need for food of the country's rising population.

Automation in Space-related activities: Data mining and machine learning techniques of AI are used to collect data from space. In the future, intelligent AI-based software will solve and draw valuable patterns and information from big data.

Automated security systems: We think stopping cybercrime will be easier if the system software is safe and based on AI. National security is one of any country's significant

challenges and most sensitive issues. If robust and intelligent software is implemented, it will soon strengthen the country's defense. e.g., the concept of a robotic army with intelligent systems installed can save the soldier's life, and many more things will be secured.

Conclusion

AI is used in SE in ways like the above literature review and the experimental implementation of an AI-based model for predicting software bugs. We have shown numerous artificial intelligence techniques that have made implementation easy and efficient in SE. We have also examined the datasets used for AI models and our experiments. This manuscript puts some essential applications on a common platform, like software defect prediction using AI-based classification algorithms. Our in-depth analysis shows that the development of AI has opened new research domains and challenges in SE for researchers and scientists to use to solve problems. We conclude that there will be a wide range of ways that AI-based techniques can be used in SE in the future, with some new and advanced hybrid algorithms. This will help in understanding and decision-making for future research.

Conflict of interest

The authors declare no potential conflict of interest regarding the publication of this work. In addition, the ethical issues including plagiarism, informed consent, misconduct, data fabrication and, or falsification, double publication and, or submission, and redundancy have been completely witnessed by the authors.

Funding

The author(s) received no financial support for the research, authorship, and/or publication of this article.

References

Bennett, K. H., & Rajlich, V. T. (2000, May). Software maintenance and evolution: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (pp. 73-87).

Bibi, S., Tsoumakas, G., Stamelos, I., & Vlahavas, I. P. (2006, March). Software Defect Prediction Using Regression via Classification. In *AICCSA* (pp. 330-336).

Briand, L. C. (2008, August). Novel applications of machine learning in software testing. In *2008 The Eighth International Conference on Quality Software* (pp. 3-10). IEEE.

Briand, L. C., Bunse, C., & Daly, J. W. (2001). A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs. *IEEE Transactions on Software Engineering*, 27(6), 513-530.

Falcini, F., Lami, G., & Costanza, A. M. (2017). Deep learning in automotive software. *IEEE Software*, 34(3), 56-63.

Gharehchopogha, F. S., & Khalifehlou, Z. A. (2012). A new approach in software cost estimation using regression based classifier. *Global Journal on Technology*, 2.

Horgan, J. R., London, S., & Lyu, M. R. (1994). Achieving software quality with testing coverage measures. *Computer*, 27(9), 60-69.

Jalote, P. (2012). An integrated approach to software engineering. Springer Science & Business Media.

Kim, S., Zhang, H., Wu, R., & Gong, L. (2011, May). Dealing with noise in defect prediction. In *2011 33rd international conference on software engineering (ICSE)* (pp. 481-490). IEEE.

Kuhn, D. R., Wallace, D. R., & Gallo, A. M. (2004). Software fault interactions and implications for software testing. *IEEE transactions on software engineering*, 30(6), 418-421.

Li, H., & Lam, C. P. (2007). Software test data generation using ant colony optimization. *International Journal of Computer and Information Engineering*, 1(1), 137-140.

Li, J., He, P., Zhu, J., & Lyu, M. R. (2017, July). Software defect prediction via convolutional neural network. In *2017 IEEE international conference on software quality, reliability and security* (QRS) (pp. 318-328). IEEE.

Malhotra[1], R., & Chug, A. (2012). Software maintainability prediction using machine learning algorithms. *Software engineering: an international Journal (SeiJ)*, 2(2).

Mustaqeem, M., & Saqib, M. (2021). Principal component based support vector machine (PC-SVM): a hybrid technique for software defect detection. *Cluster Computing*, 24(3), 2581-2595.

Nassif, A. B., Capretz, L. F., & Ho, D. (2012, December). Estimating software effort using an ANN model based on use case points. In *2012 11th International Conference on machine learning and applications* (Vol. 2, pp. 42-47). IEEE.

Navarro-Almanza, R., Juarez-Ramirez, R., & Licea, G. (2017, October). Towards supporting software engineering using deep learning: A case of software requirements classification. In *2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT)* (pp. 116-120). IEEE.

Parizi, R. M., & Ghani, A. A. A. (2010, May). Towards Automated Monitoring and Forecasting of Probabilistic Quality Properties in Open Source Software (OSS): A Striking Hybrid Approach. In *2010 Eighth ACIS International Conference on Software Engineering Research, Management and Applications* (pp. 329-334). IEEE.

Pressman, R. S. (2010). A practitioner's approach. Software Engineering, 2, 41-42.

Sadiku, M. N. O., Zhou, Y., & Musa, S. M. (2018). Natural language processing in healthcare. *International Journal of Advanced Research in Computer Science and Software Engineering*, 8(5), 39-42.

Saini, D. (2016). Applications of various artificial intelligence techniques in software engineering. *International Journal for Research in Emerging Science and Technology*, 3(3), 25-33.

Siddiqui, T., Mustaqeem, M., Athar, S., Khan, N., & Hasan, S. (01 2021). *Impact Analysis of Machine Learning Techniques in Software Engineering. GIS-Zeitschrift Fü Geoinformatik*, 8.

Srivastava, P. R., & Baby, K. (2010, December). Automated software testing using metahurestic technique based on an ant colony optimization. *In 2010 international symposium on electronic system design* (pp. 235-240). IEEE.

Strnad, D., & Guid, N. (2010). A fuzzy-genetic decision support system for project team formation. *Applied soft computing*, 10(4), 1178-1187.

Yalla, P., & Sharma, N. (2015). Integrating natural language processing and software engineering. *International Journal of Software Engineering and Its Applications*, 9(11), 127-136.

Bibliographic information of this paper for citing:

Mustaqeem, Mohd; Siddiqui, Tamanna; Khan, Najeeb Ahmad & Kumar, Deepak (2023). In-Depth Analysis of Various Artificial Intelligence Techniques in Software Engineering: Experimental Study. *Journal of Information Technology Management*, 15 (3), 162-181. https://doi.org/10.22059/jitm.2023.93632