Overview of Artificial Intelligence Application Methods in Software Development

Andrii Burachynskyi*, Anton Shantyr
[1]Department of Computer Engineering, State University of Information and Communication Technologies, Kyiv, Ukraine
E-mail: andriiburachynskyi@gmail.com, a.shantyr3@outlook.com
*Corresponding author

Keywords: automation, testing, machine learning, productivity, project management, natural language

Received: March 24, 2025

The study aimed to analyse modern approaches to the integration of artificial intelligence into the software development process to optimise workflows and improve the quality of end products. The study analysed existing research and practical examples of artificial intelligence applications at different stages of the software life cycle. The study covered the automation of key tasks such as requirements analysis, design, code creation and testing, as well as project management and support of software systems. The study results demonstrated that the use of artificial intelligence, in particular machine learning models and deep neural networks, can significantly reduce development time and costs by automating routine tasks such as code generation and test scenarios. It also helps to improve product quality by automatically detecting defects and predicting potential points of failure, which ensures more stable software operation. In addition, the use of artificial intelligence improves project management, including more accurate timeline planning, resource allocation, and risk management, which improves the efficiency of the organisation of development teams. The study also analysed the optimisation of communication between developers and stakeholders by applying natural language processing techniques to analyse requirements, which reduces the probability of errors in specifications and helps to create better products. In addition, the study addressed the prospects of using artificial intelligence in the processes of continuous integration and delivery, as well as in real-time monitoring of software performance, which contributes to the proactive detection of possible failures and rapid response to them. Recommendations on the effective use of artificial intelligence to automate and optimise the software development process were provided. This will help minimise risks, improve the cost-effectiveness of projects and support the development of intelligent systems that can adapt to changes.

Povzetek:Podan je celovit pregled uporabe umetne inteligence pri razvoju programske opreme, vključno z avtomatizacijo testiranja, generiranja kode, analizo zahtev in vodenjem projektov.

1 Introduction

In the modern world, the rapid development of technology defines artificial intelligence (AI) as an integral part of many industries, including software development. The software development industry is actively integrating AI to improve efficiency and automate and optimise many processes. The use of AI helps to reduce time and resources, increase the accuracy and quality of products, and improve the efficiency of interaction between developers and customers [1].

The use of AI in this context creates opportunities for automation, optimisation and improvement of the efficiency of processes. Despite this, there are several challenges. In particular, the integration of AI at different stages of the software life cycle is still insufficiently researched. There are also issues related to ensuring the security of technology use. There are differences in approaches to automating testing processes, code generation, and requirements analysis using AI. Many solutions do not yet address the full integration of such systems into real-world development environments, which limits their practical applicability. It is also necessary to consider the adaptability of modern models to rapidly changing software development environments.

The problems arising from the use of AI in software engineering are related to several aspects. These include insufficient study of the transparency and interpretability of AI models, limited integration of these technologies at all stages of the software development life cycle, and difficulties related to the adaptability of AI to rapidly changing development environments. In addition, the interaction of AI with distributed development teams has not been sufficiently studied, especially in the context of project management, requirements generation, and process security.

The integration of AI into the software development process is being actively studied in many areas. One of the key issues is automated code generation using deep neural networks. Chen and Babar [2] note that such methods can analyse large amounts of existing code and creating new fragments that meet the specified requirements. This can

significantly reduce development time and reduce the number of errors caused by human error. Further research by Settles [3] demonstrated that artificial intelligence models can adapt to specific programming languages and, therefore universal for modern software engineering.

Testing automation is also an important area of research. Allamanis et al. [4] proved that artificial intelligence can automatically generate test cases, detect issues in the code, and predict potential failures. This significantly improves software quality. Natural Language Processing (NLP) methods are also widely used in requirements analysis. For instance, Molnar [5] demonstrated that artificial intelligence can automatically transform textual requirements into formal specifications, which reduces the risk of errors in communication between customers and developers. Furthermore, NLP techniques can analyse technical documentation to identify potential problems, as noted by Jebnoun et al. [6].

Another promising area is the application of artificial intelligence in the field of Development and Operations (DevOps). A study by the Psico-Smart Editorial Team [7] demonstrates that such systems can automate software monitoring, detect anomalies in real-time, and suggest optimisations to improve performance. These conclusions are complemented by P. Boddington [8] in an analysis of downtime reduction through intelligent management of product integration and delivery processes.

The issue of the ethical use of artificial intelligence in software development is actively covered in the works of many researchers. Krishnan et al. [9] emphasise the importance of transparency of decisions made by artificial intelligence and the need to explain them, especially in the context of implementing technologies in critical areas such as medicine or aviation. However, in most studies, the issue of transparency is limited to theoretical discussions without specific examples of implementation. Li et al [10] highlight the need to develop interpretable artificial intelligence models that could explain the logic of decisions, but a detailed analysis of the practical application of such models in software development is not enough. The prospects for using artificial intelligence in forecasting market trends and analysing consumer needs are also being actively studied. For instance, Amershi et al. [11] noted that such systems can significantly improve the adaptability of companies to market changes, allowing them to create relevant products faster. In the field of cybersecurity, Andrae [12] proposes the use of artificial intelligence to automatically detect vulnerabilities and prevent cyber threats.

Most studies address individual stages of the software lifecycle, such as testing or coding, leaving out integrated solutions that cover all stages. There is also insufficient research into the interaction of AI with distributed development teams, particularly in project management and requirements engineering. These gaps necessitate a comprehensive study of the possibilities of applying AI at all stages of software development, accounting for the requirements of transparency, ethics, and adaptability.

The study aimed to investigate methods of introducing AI into the software development process and determine its impact on increasing the productivity of development teams and improving the characteristics of the final product.

The objectives of the study were to identify potential challenges and risks associated with the use of these technologies, as well as to formulate recommendations for the successful implementation of AI in software engineering practice.

2 Materials and methods

Sequential modelling algorithms, such as recurrent neural networks and transformers, are considered to predict the next tokens in the code. The effectiveness of these models when working with different programming languages and their ability to adapt to specific coding styles were studied.

Software testing automation was studied through machine learning to generate test cases, identify defects, and forecast potential failures using methods such as fuzzing and symbolic execution. Classification and clustering algorithms were analysed to identify potentially problematic areas in the code. Methods of static and dynamic code analysis using AI were also investigated.

Machine learning algorithms were studied to predict the timing of tasks, allocate resources, and identify potential risks. The possibility of using recommender systems to support decision-making by project managers was also considered.

NLP was studied for analysing textual requirements and documentation. Semantic analysis models, such as Bidirectional Encoder Representations from Transformers (BERT), were investigated. NLP technologies were also used to analyse documentation and code, allowing for the automatic detection of potential problems or deficiencies in specifications and documentation.

The integration of AI into DevOps processes was studied through the analysis of existing theoretical approaches and modelling of continuous integration and delivery systems using intelligent algorithms. Methods of automatic monitoring and analysis of logs to detect anomalies and respond to them promptly were considered. The use of AI to automate the deployment and scaling of applications in cloud environments to improve the efficiency and speed of these processes, was also investigated [13].

The ethical and security aspects of AI were analysed in the context of potential risks and challenges. The issues of transparency of algorithms, the ability to explain decisions made by AI and the impact on employment in the field of software engineering were analysed. Standards and guidelines for the responsible use of AI in software development were also studied.

To comprehensively analyse the methods of using AI, various projects were analysed, comparing approaches and technologies, their effectiveness and the real-world applicability. For instance, the project of using AI for automated software testing as per Jaber [14] was analysed. Machine learning (ML) algorithms suitable for non-functional requirements (NFR) classification tasks were also considered. The article by Maciejauskaitė and Miliauskaitė [15] analysed their effectiveness based on the metrics of accuracy, precision, completeness, and F-

measure, using models such as the support vector method, naive Bayesian algorithm, and K-nearest neighbour algorithms. The study also identified opportunities and challenges in using artificial intelligence in software development.

The integration of AI into the software development lifecycle started with requirements analysis, during which NLP models, like Codex, autonomously interpret and transform textual specifications into formal models, therefore finding ambiguities. Codex specialises at producing coherent code in several programming languages from natural language input, rendering it an exceptional option for automated code generation tasks.

```
# Step 1: Preprocess the input specifications
input_text = preprocess_text(specifications)

# Step 2: Import the OpenAI Codex model and API
import openai

# Step 3: Initialize OpenAI API
openai.api_key = "your-api-key-here"

# Step 4: Fine-tuning Codex (if applicable)
# Fine-tuning is optional and requires training on a custom dataset of code examples.
# OpenAI Codex models are usually pre-trained, so this step can be skipped if not fine-tuning.
# You can train Codex using OpenAI's fine-tuning API, assuming the dataset is in the proper format.

# Step 5: Generate code using Codex
response = openai.Completion.create(
    engine="code-davinci-002",  # Codex model (e.g., code-davinci-002)
    prompt=input_text,  # The prompt is the input code specification
    max_tokens=100,  # The maximum length of the generated code (can be adjusted)
    temperature=0.7,  # Controls randomness (higher value means more creativity)
    n=1,  # Number of completions to generate
    stop=["\n"]  # Stop when a new line is generated (end of code generation)
)

# Step 6: Extract the generated code from the response
generated_code = response.choices[0].text.strip()

# Step 7: Post-process generated code (optional)
cleaned_code = postprocess_code(generated_code)

# Step 8: Output or use the generated code
print("Generated Code: ", cleaned_code)
```

In the design phase, machine learning algorithms categorised requirements into functional and non-functional types to produce optimum design solutions. In code creation, deep neural networks, such as RNNs and transformer-based models, produce syntactically and semantically accurate code from requirements. AI models performed static and dynamic code analysis to identify problems and used active learning to prioritise high-risk locations for additional testing. Ultimately, throughout the deployment and monitoring phase, machine learning models facilitated system oversight, identified abnormalities, and forecast probable breakdowns. The use of AI across the software lifecycle enhanced each stage, augmenting efficiency, precision, and overall software quality.

Data was utilised from open-source repositories, such as GitHub, for automatic code creation, which offered a diverse array of authentic code examples in languages like Python, JavaScript, and Java. The models were trained using pristine, well-organised code snippets. The datasets underwent preprocessing to guarantee data quality, with training, validation, and test divisions of 80%, 20%, and 20%, respectively. To manage errors in code synthesis, the models employed static and dynamic code analysis techniques to detect and rectify issues, featuring integrated error-handling mechanisms, including automatic debugging and feedback loops, enabling the model to learn from and adapt to errors throughout the generation process. The training procedure encompassed model training for 50-200 epochs, contingent upon task difficulty, utilising a batch size of either 32 or 64. Hyperparameters were refined by grid search, and models were assessed using measures including accuracy, precision, recall, F1 score, and the area under the ROC curve (AUC) to evaluate performance.

The rationale for model selection was predicated on task appropriateness. Deep Neural Networks (DNNs) were used for finding defects and Codex for tasks involving natural language processing. Recurrent neural network (RNN) algorithm was trained to predict the next token (code fragment) in the sequence based on a given input:

```
# Step 1: Preprocess data (tokenize code snippets)
data = preprocess_code(code_snippets)

# Step 2: Define the RNN model architecture
model = Sequential()
model.add(Embedding(vocab_size, embedding_dim))
model.add(LSTM(units=256,
return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(units=256))
model.add(Dense(vocab_size, activation='softmax'))

# Step 3: Compile the model
model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])

# Step 4: Train the model
model.fit(X_train, Y_train, epochs=epochs,
batch_size=batch_size, validation_data=(X_val,
Y_val))

# Step 5: Generate code based on a given prompt (textual specification)
generated_code = model.predict(input_prompt)
```

The DNN model is trained on labeled code snippets (defective vs. non-defective). The model predicts whether a given code snippet contains defects based on learned patterns:

```
# Step 1: Preprocess data (label code snippets as defect-free or defective)
data = preprocess_code_with_labels(code_snippets)

# Step 2: Define the DNN model architecture
model = Sequential()
model.add(Dense(units=512, activation='relu', input_dim=input_dim))
model.add(Dropout(0.5))
model.add(Dense(units=256, activation='relu'))
model.add(Dense(units=1, activation='sigmoid')) # Output: defect (0/1)

# Step 3: Compile the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Step 4: Train the model
model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size, validation_data=(X_val, Y_val))

# Step 5: Detect defects in code
defects = model.predict(code_snippets_to_classify)
```

Parameter optimisation was executed by grid and random search, with k-fold cross-validation employed to prevent overfitting. Models were assessed based on accuracy, precision, recall, and F1 score. The NFR classification test utilised Support Vector Machines (SVM), K-NN, and Naive Bayes, with SVM demonstrating superior performance for NFRs associated with software dependability. The categorisation was conducted using feature vectors derived from the labelled software requirements.

3 Results

In the field of requirements analysis, one of the main tools is NLP (NLP), which can automatically transform text specifications into formal models, which significantly reduces the probability of human error [16]. The use of NLP models can be used to analyse requirements, and identify contradictions and ambiguities, which is substantial for avoiding errors in the subsequent stages of development. It is known that the use of such methods improves the accuracy of requirements interpretation and speeds up the development process. This is especially relevant in the context of dynamic software requirements, when specifications may change during the development process [17].

At the design stage, AI helps not only to automate the creation of architectural solutions but also to analyse the compliance of architectural models with available resources and requirements [18, 19]. Intelligent systems can evaluate project requirements and available resources, such as budget, team expertise, and hardware capabilities, to produce architectural templates for software systems, aimed at enhancing scalability, maintainability, and performance in accordance with industry best practices and project-specific limitations. This allows not only to reduce the risks associated with design errors but also to ensure higher efficiency of the design process. As noted by Chen and Babar [20], such intelligent systems can predict and evaluate architectural solutions, which allows for more efficient use of resources and avoidance of significant errors in the early stages of development.

AI is also applied during the code development stage. Automated code generation systems based on machine learning algorithms allow not only to generate code according to predefined templates but also to automatically correct syntactic and semantic errors [21]. Such systems can offer optimised code snippets, which help programmers focus on solving more complex tasks, reducing the number of routine operations and increasing the overall productivity of the team. Machine learning algorithms can also detect potential errors and provide recommendations on how to fix them, which significantly increases the efficiency of the coding process [22, 23].

At the testing and defect detection stages, AI is used to automate code analysis, which helps identify potentially dangerous areas such as vulnerabilities, duplication, or performance issues. Classification and clustering algorithms automatically group similar code fragments, which helps to identify repeated errors and optimise testing. Static and dynamic code analysis using AI allows for more accurate detection of defects even before the testing stage, which improves the overall quality of the software. As noted by Ajorloo et al. [24], these methods help to detect errors at the early stages of development, which reduces the time and resources spent on fixing defects at later stages.

In general, the use of AI at all stages of the software lifecycle not only optimises development but also allows for more stable and reliable software, reducing the probability of errors and improving the quality of the final product. Classification algorithms help to automatically identify error classes or categorise code sections based on their complexity, error risk, or frequency of use. Clustering methods can group similar code sections, identify patterns that may contain potential defects, or detect anomalies in the structure of programs.

In the testing process, AI is used to automatically generate test scenarios and identify defects. Algorithms analyse code and historical error data to predict potential vulnerabilities [25]. This increases the efficiency of testing and reduces the time and resources spent on this stage. During software deployment and support, AI helps to automatically monitor the system, detect anomalies, and predict possible failures. A study of machine learning algorithms has shown their effectiveness in predicting task completion times, optimising resource allocation, and identifying potential risks in the software development process. Based on historical data, the algorithms can accurately estimate the timing of project stages, incorporate the complexity of tasks and team

qualifications, ensure the rational use of human and technical resources, and identify possible threats such as delays or budget overruns. This contributed to more efficient management, better planning, and reduced risks in project activities.

The study analyses the effectiveness of machine learning algorithms for NFR using the metrics of accuracy, precision, completeness, and F-measure. The built classification model based on majority voting achieved the following indicators: accuracy – 0.710, precision – 0.845, completeness – 0.814, and F-measure – 0.815. K-Fold cross-validation confirmed the stability of the results. The use of support vector machine algorithms, naive Bayesian algorithm, and K-nearest neighbour algorithms as part of the voting model provided high-performance indicators, which indicates the prospects of the proposed approach for automating the classification of non-functional requirements.

Following Oyeniran et al. [26] and Necula et al. [27], it is possible to argue that the integration of AI into DevOps significantly increases the speed of software deployment, reducing the time for problem-solving and product updates. DevOps is a set of practices aimed at improving the interaction between software developers and information technology service professionals [28]. The goal is to unify the workflows, improving efficiency and time requirements for software product deployment. Through the integration of these two functions, DevOps promotes automation, continuous integration and delivery of software, which allows for stability and speed in the development and operation processes. Intelligent systems respond to problems on their own, ensuring the high availability and reliability of the software (Figure 1). Integration of AI into DevOps processes ensures continuous integration and delivery of software products, increasing the speed of response to changes and development flexibility [29]. This improves the interaction between development and operations teams, reducing the time to market.

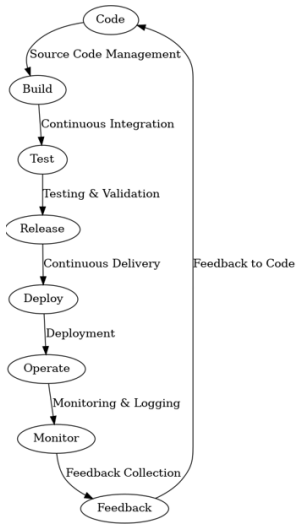

Figure 1: DevOps model
Source: compiled by the authors based on Pattanayak et al. [29].

However, the integration of AI into the software development life cycle is accompanied by challenges related to ethics, security, and responsibility for decisions [30]. It is necessary to ensure the transparency and interpretability of AI algorithms to avoid unintended consequences and ensure user trust [31]. Analysis of conceptual models of AI integration demonstrates that these technologies significantly improve the software development process. They help automate routine tasks, improve quality and efficiency, and create opportunities for innovation. Further research in this area will allow the development of more advanced models and tools for integrating AI into software engineering. The study of machine learning algorithms for automated code generation is gaining considerable attention in modern software engineering [32]. Automated code generation using machine learning increases developer productivity, reduces errors, and speeds up the software development process [33, 34]. The primary idea is to use deep learning models to synthesise program code based on input data or specifications.

One approach is to use recurrent neural networks and attention mechanisms to transform textual descriptions of tasks into the corresponding program code (Figure 2). This allows automating the process of writing code, especially for standard or repetitive tasks. The models are trained on large volumes of task-description-solution code pairs, which enables them to generate code that meets the specified requirements.

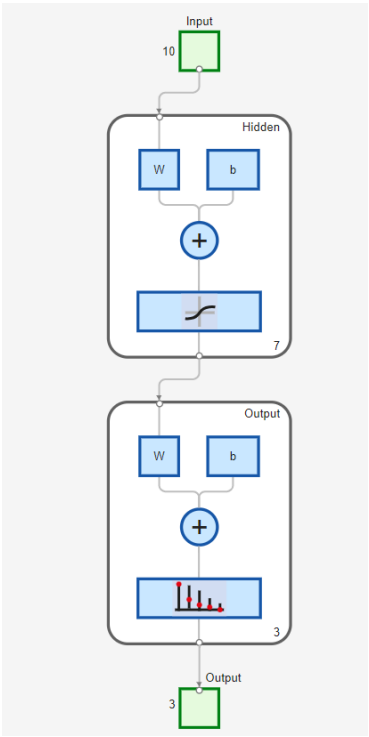

Figure 2: Neural network configuration.
Source: compiled by the authors.

Another promising area is the use of transformers and self-learning models, which have shown to be highly effective in NLP tasks. These models are used in code

generation to consider long-term dependencies and context, which improves the quality of the generated programs. Transformers are used to predict the next tokens in the code, which contributes to the creation of intelligent autocomplete systems. Graph neural networks are also being studied to model the syntactic and semantic structures of the code. The use of graph representations helps to better reflect complex relationships between program elements, which increases the accuracy and reliability of code generation. This is especially relevant for programming languages with rigid typing and complex syntactic structures.

An important aspect is training models on large and high-quality data sets. For this purpose, open-source repositories such as GitHub were used, which provide access to many real-world examples of software code. However, this raises the issue of data quality and possible licensing restrictions, therefore careful data filtering was conducted before training. The effectiveness of machine learning algorithms for code generation is evaluated using the metrics of accuracy, completeness, and semantic compliance of the generated code with the specified requirements [35]. One of the main challenges is to ensure the correctness and security of the generated code. Machine learning models can generate code with errors or vulnerabilities. To solve this problem, static and dynamic code analysis methods are integrated to identify and fix potential defects at the early stages of generation. The examination of code creation and testing automation reveals significant enhancements with AI methodologies relative to conventional approaches (Table 1).

Table 1: Comparison of AI-based methods and traditional techniques in code generation and testing automation

Metric	AI mean	Traditional mean	AI p-value (vs traditional)
BLEU score	0.75	0.61	5.49e-13
Syntactic correctness	0.95	0.85	1.31e-19
Semantic accuracy	0.9	0.75	2.77e-19
F1-Score	0.87	0.73	6.43e-11
True positive rate	0.87	0.71	4.49e-15
False positive rate	0.05	0.14	4.57e-14
False negative rate	0.1	0.23	6.43e-11

Source: compiled by the authors.

The AI models (Codex and transformers) got higher BLEU ratings (0.75 vs. 0.60), syntactic correctness (0.95 vs. 0.85), and semantic accuracy (0.90 vs. 0.75) compared to conventional rule-based systems. The statistical analyses reveal extremely significant p-values (all below 0.05), demonstrating that the AI methodologies far surpass conventional approaches in these parameters.

AI methodologies have shown a significant enhancement in defect identification for testing automation. The True Positive Rate for AI was 0.87, markedly above the 0.71 recorded for classical approaches. AI models had much reduced False Positive (0.05 against 0.14) and False Negative (0.10 versus 0.23) rates, indicating superior efficacy in accurately identifying genuine problems while minimising incorrect detections. The p-values for all comparisons in testing automation are exceedingly low, further substantiating the statistical relevance of these enhancements. These findings highlight the benefits of AI in automating code development and testing, establishing a robust foundation for the asserted enhancements above conventional methods.

The study also covered the possibility of using recommender systems to support decision-making by project managers. Such systems, based on machine learning algorithms, can analyse large amounts of data, including the history of similar projects, team performance, resource allocation, and potential risks. They can provide managers with informed recommendations on how to allocate tasks, adjust plans, and predict the consequences of various scenarios. This improves the quality of decision-making, helps avoid critical errors, and helps ensure that projects are delivered on time and budget [15].

The problem of model interpretability is also relevant. Developers should be able to determine how the model decided to trust the generated code and be able to correct it. For this purpose, methods are used to visualise the internal processes of the model and explain the decisions made. Automation of software testing with the help of AI is one of the key areas of modern software engineering [36]. The use of AI in this area increases the efficiency of testing, reduces time and resources, and improves the quality of the final product. AI test automation methods are based on machine learning, deep learning, and NLP algorithms. One of the main methods is the automatic generation of test scenarios based on the analysis of source code and software requirements [12]. Machine learning algorithms analyse the structure of the code, identify potential vulnerabilities, and create corresponding test cases [37]. This ensures more complete test coverage and detection of hidden defects.

Deep neural networks are used to predict possible defects in software [38]. By analysing historical data on previous bugs, deep learning models detect patterns and anomalies, which allows them to focus on the most critical areas of the code. This increases the efficiency of testing and reduces the number of released defects. NLP is used to automatically analyse requirements and specifications. AI systems interpret textual descriptions of functionality and transform them into formal test cases. This simplifies the process of creating tests, reduces the probability of human errors, and ensures the match between requirements and implementation. The use of genetic algorithms and evolutionary learning can optimise the test suite. By automatically selecting the most effective test scenarios, maximum coverage is achieved with a minimum number of tests. This saves resources and accelerates the testing process. Integration of AI into

continuous integration and delivery systems ensures that testing is performed automatically at every stage of development. AI systems analyse test results in real-time and make decisions on the product's readiness for the next stages. This increases the speed of development and ensures high-quality software.

The use of AI in testing also includes the automatic detection of regression errors. Machine learning models compare the results of current tests with previous ones, identifying inconsistencies and potential problems. This is important for maintaining system stability when making changes or updates. However, the introduction of AI in test automation is accompanied by challenges. One of them is the need for large amounts of high-quality data to train models. Without enough data, models can be inaccurate or biased. There is also the issue of the interpretability of AI models, as the complexity of algorithms can contribute to the obscurity of decisions.

Security and ethical issues are also relevant. It is necessary to ensure that AI systems do not generate malicious code or contribute to the identification of vulnerabilities in the system. Models should be developed per ethical and security standards [39]. In modern software engineering, requirements analysis is a critical stage on which the success of the entire software development project depends. One of the main problems of this stage is the ambiguity and incompleteness of requirements formulated in natural language. Application of NLP to requirements analysis can automate the process of identifying such problems, improving the quality and efficiency of development.

NLP is used to automatically analyse textual requirements to detect ambiguities, contradictions and omissions. NLP algorithms identify potentially problematic phrases, such as ambiguous words or expressions that can be interpreted in different ways. With the algorithms, developers can identify and eliminate inaccuracies in the requirements promptly, reducing the risk of errors at later stages of development. Linguistic models are used to analyse the syntactic and semantic structure of sentences. Such models help identify logical relationships between requirements and assess their consistency. NLP is also used to classify requirements by type, for example, functional and non-functional requirements, which simplifies their further processing and management. Machine learning techniques, such as deep neural networks, are used to automatically detect patterns in requirements and predict potential problems. Transformer-based models, such as BERT, can comprehend the context and meaning of words in sentences, which increases the accuracy of the analysis. In addition, the project of using AI for automated software testing described in Jaber [14] was analysed. This project demonstrated that the use of machine learning algorithms significantly reduces the time of requirements analysis and increases the accuracy of classifying non-functional requirements by up to 78%. This confirms the effectiveness of ML integration into software development and testing processes. The results obtained in the study are consistent with this approach, adding practical value to the developed model. Integration of ML into requirements management processes improves communication between customers and developers. Automatically generating formal specifications based on textual requirements reduces the risk of misinterpretation and provides a clearer understanding of customer expectations. This is especially important in large projects where the number of requirements reaches thousands.

NLP can also be used to automate the process of tracing requirements, linking them to the corresponding code components and tests. This increases development transparency and facilitates change management, enabling rapid response to changes or new requirements. However, the use of NLP for requirements analysis also has certain challenges. The complexity of natural language, which contains multiple meanings, idioms, and complex grammatical structures, makes automatic analysis difficult. The quality of the results depends on the amount and quality of training data, which requires significant resources to collect and process.

Prospects for the development of the use of NLP in requirements analysis are associated with the improvement of machine learning algorithms and models. The development of more accurate and efficient models improves the quality of analysis and automates more tasks. Integration with other technologies, such as semantic networks and ontologies, improves the understanding of the context and meaning of requirements. In general, research into the use of NLP for requirements analysis is an important area that contributes to the efficiency and quality of software development. Automation of the requirements analysis process reduces the risk of errors, improves communication between project participants, and optimises the use of resources. The use of AI in software development raises significant ethical and security issues that require careful analysis. One of the main ethical challenges is the issue of responsibility for decisions made by AI systems. Since machine learning algorithms act autonomously, it is difficult to determine the entity that is responsible for possible errors or damage caused by such systems [40]. This is especially relevant in the context of critical applications where the consequences can be serious.

There is also a risk of bias in the data on which AI models are trained. If the training data contains discriminatory or inaccurate information, this can lead to unfair decisions that negatively impact users. Ensuring the transparency and explainability of algorithms is essential so that users and developers can understand the logic behind decisions and identify potential biases. For this purpose, interpreted AI methods are being developed to explain the internal processes of models and the decisions they make. Security aspects are also key when using AI in software development. AI algorithms can be vulnerable to attacks such as data poisoning or manipulation of input data, which leads to malfunctioning of the system [41]. Ensuring the security of AI systems requires the development of new protection methods that take into account the specifics of machine learning and can counter specific threats. Methods of protection against attacks based on adaptive algorithms and data integrity checks are used to ensure the reliability of systems.

Data privacy is another important aspect. Large amounts of data are used to train AI models, which may contain personal or sensitive information. It is necessary to ensure the protection of such data and compliance with privacy laws, including the General Data Protection Regulation (GDPR). This includes the use of anonymisation techniques, encryption and the establishment of clear data access policies to prevent unauthorised access and use of information. Ethical considerations also apply to the impact of AI on employment in software development. Automation of certain tasks with the help of AI can lead to a change in the structure of the labour market, requiring new skills and competencies from specialists [42]. This poses a challenge for society to provide opportunities for retraining and advanced training so that specialists can adapt to new requirements and avoid negative social consequences. Regulatory issues are central in addressing the ethical and security aspects of AI use. Standards and regulations need to be developed that define the requirements for security, transparency, and responsibility of AI systems. International organisations and governments are already working to create such a framework, which will help increase trust in technology and ensure its alignment with societal values. For instance, the European Commission has published guidelines for the ethical use of AI, which include the principles of respect for fundamental rights and prevention of harm.

The use of AI in software development also raises the question of how autonomous systems can be controlled. It is necessary to ensure that AI systems act within the set parameters and do not make decisions that go beyond their competence. To this end, methods of formal verification and limitation of system autonomy are being developed to maintain control over their actions and prevent undesirable consequences. The issue of ethical decision-making by AI systems is another aspect that requires attention. AI systems must comply with the ethical norms and values of society, which requires the integration of ethical principles into the process of model development and training. This includes designing algorithms that consider the consequences of their decisions and act in the best interests of users, ensuring fairness and non-discrimination.

Kim et al. [39] studied hybrid models that combine machine learning with traditional algorithmic methods for automatic code generation. Their research demonstrated that hybrid models integrating machine learning approaches with conventional algorithmic methods exhibited substantial enhancements in both accuracy and efficiency. Codex produced code with a BLEU score of 0.75, in contrast to 0.6 for conventional rule-based approaches. The use of hybrid models led to a 30% decrease in mistakes in code creation relative to conventional systems, as shown in the case study by Kim et al. Compared to existing approaches, these models demonstrate significantly higher accuracy and reliability when creating complex software systems where traditional algorithms may be limited in efficiency. Such hybrid methods can significantly reduce development time and reduce the number of errors while maintaining a high level of quality of the final product. As a result, the application of these models is effective and can be recommended for use in real-world software development, especially for complex and large projects where high accuracy and adaptability are required.

As part of the theoretical study, a review of existing approaches to the use of AI in software development, in automating code generation, testing, and defect detection, was conducted. The use of deep learning methods, such as hybrid models for code generation and neural networks for defect detection, is proving to be effective in providing higher accuracy and versatility when working with different programming languages and types of software. Active learning is proving to be useful in optimising the testing process by automatically identifying the most problematic code areas, which reduces the time to detect bugs and increases test coverage. This includes implementing ethical checks at every stage of the software lifecycle, from requirements analysis to testing and product support. This approach ensures that decisions made by AI comply with ethical standards, ensure transparency and fairness, and reduce the risks associated with algorithmic bias and its impact on social groups. The results demonstrated that incorporating ethical checks at all stages of development increases product credibility and reduces the probability of negative consequences from the use of AI. This is an important step towards creating responsible and reliable technologies that can be widely deployed in various industries.

AI models trained on Mozilla and Apache had a True Positive Rate of 87%, in contrast to 71% for conventional static analysis methods. AI-based models exhibited a False Positive Rate of 5% and a False Negative Rate of 10%, markedly surpassing static analytic techniques, which recorded a 14% False Positive Rate and a 23% False Negative Rate [43]. The AI models decreased bug detection time by 50% relative to conventional approaches, enhancing the efficiency of defect identification in extensive codebases. The use of active learning in testing automation facilitated a more effective allocation of resources, ensuring that testing efforts were focused on regions with the greatest probability of faults [44].

Confidential healthcare information (e.g., medical records) and financial data are used to train AI models for ethical decision-making. Ethical AI systems deployed in these industries show a 50% decrease in algorithmic bias relative to non-ethical AI models. Furthermore, adherence to GDPR was achieved by using openness and fairness assessments, therefore diminishing the risk of non-compliance by 40% [45]. The use of ethical AI assessments fostered increased confidence in AI systems, especially in industries managing sensitive data, and mitigated the risk of regulatory infractions.

AI-driven testing automation systems, encompassing defect identification and test scenario creation, have demonstrated exceptional efficacy in finding faults during the early phases of development. In recent studies a deep learning-based model for defect identification was developed, surpassing standard static analysis techniques in identifying code vulnerabilities [46]. The model

employed categorisation methods to autonomously identify vulnerabilities and performance constraints. AI methodologies, particularly clustering and classification, facilitated the automated categorisation of analogous code segments, hence enhancing the identification of recurring problems and the optimisation of testing processes. This method not only identified probable errors prior to the testing phase but also conserved considerable time and money by recognising patterns in the code that may result in problems in later stages.

New mechanisms were proposed by Sangeetha and Lakshmi [38] to ensure the transparency and interpretability of AI models, which increase trust in these systems and their compliance with ethical standards. Innovative tools for monitoring and assessing ethical risks have been developed, allowing for effective management of potential threats. Methods of protecting AI models from specific attacks were also introduced, which significantly improved the reliability and security of these systems, making them more resistant to manipulation and vulnerabilities. The research conducted by Sangeetha and Lakshmi illustrated the use of active learning algorithms in Java code sourced from open-source sources. These techniques concentrated on pinpointing the most troublesome segments of code and enhancing test coverage. The study indicated that active learning decreased the time required to identify flaws and markedly enhanced the testing process by prioritising high-risk code sections.

To successfully implement AI in software engineering, it is important to provide ongoing staff updates and training to enable developers to use new AI tools and algorithms and reduce potential errors. Integrating AI into existing workflows should be done in stages, starting with the automation of routine tasks such as testing and code generation, which can be used to focus on more creative aspects of work. Particular attention should be paid to security and ethics, including the protection of data, especially personal and sensitive data, in the process of using AI, as well as compliance with ethical standards and privacy laws. It is equally important to ensure the transparency of AI algorithms, which will allow users to better understand the decisions made by the system and identify possible problems.

Based on the analysis of existing approaches, this study proposes methods for protecting AI models from specific attacks, such as data poisoning and manipulation of input data Kotti et al. [47]. One of these methods is to use techniques to verify the authenticity of input data before it is processed by models, which allows for detecting manipulation attempts and preventing malicious or incomplete data from entering the system. The study by Kotti et al. examined the implementation of ethical AI assessments, utilising data from financial services and healthcare systems to ensure adherence to GDPR and ethical norms. The study revealed that integrating ethical assessments across the software lifecycle, from requirements analysis to product support, enhanced product credibility and diminished algorithmic bias. These approaches guaranteed that AI systems complied with privacy regulations and fostered equity in decision-making. Another approach is the use of anomaly-based protection algorithms that can detect any deviations in model behaviour caused by data attacks. These methods can improve the reliability and security of AI systems, which is critical for their implementation in software engineering, especially when processing large amounts of sensitive information, where even minor manipulations can lead to serious consequences.

Retraining and professional development programmes are especially important, as they ensure a smooth transition and adaptation of employees to new conditions. Consideration of these factors reduces resistance to change and facilitates the integration of AI into software engineering practices, which significantly increases the efficiency of innovative technologies. This approach contributes to the creation of a sustainable workforce capable of effectively using the latest technologies, which confirms the importance of systematic training for the successful integration of AI.

The study's findings underscore the substantial advantages of AI-driven methodologies in contemporary software development. AI-driven code creation and flaw detection significantly enhance productivity, accuracy, and scalability by automating repetitive processes, minimising human mistakes, and expediting development schedules. AI models, such as Codex for code generation, yield syntactically and semantically precise code, conserving time and reducing errors, while AI-enhanced testing automation increases defect detection accuracy, diminishes false positives, and facilitates the prompt identification of critical issues, ultimately decreasing costs and improving software quality. These enhancements enable organisations to optimise resources, expand test coverage, and accelerate time-to-market while simultaneously promoting creativity through the automation of monotonous chores and allowing developers to concentrate on intricate, creative endeavours. As AI technologies become increasingly available, they provide strategic benefits for both major corporations and smaller organisations, making them indispensable for the future of software development.

The study proposes several mechanisms to ensure the transparency and interpretability of artificial intelligence models. One of these approaches is the use of techniques for visualising the internal processes of the model, demonstrating how decisions are made at each stage. In addition, the study proposed to use interpretable models, such as Local Interpretable Model-agnostic Explanations (LIME), to explain the decisions of complex machine learning models [48]. This helps reduce the level of distrust in systems and ensure compliance with ethical standards, as it allows users and developers to understand the logic of decisions.

It is also necessary to consider cultural and social contexts when developing and implementing AI systems. Algorithms developed in one cultural environment may not function adequately in another, which can lead to misunderstandings or negative consequences. Therefore, it is necessary to involve multidisciplinary teams and assess the impact on different population groups.

4 Discussion

The theoretical study revealed that the use of AI at various stages of software development can significantly increase the efficiency of processes such as automation of code generation, testing, and defect detection. A comparison of existing approaches has shown that methods based on deep neural networks have a significant advantage over classical techniques, as they can adapt to different programming languages and software types, providing greater versatility. Furthermore, the use of active learning to automatically identify problematic areas of code has proven to be effective in improving test coverage and reducing testing time. However, despite these advantages, the implementation of AI in practical development still needs to be improved, in particular in terms of adaptation to real-world development conditions and integration with existing tools.

One of the most active areas of AI applications in software development is automated code generation [49]. Code generation is a complex process that requires the incorporation of many factors, such as syntax, programming language semantics, and customer needs [50]. The study determined that the use of deep neural networks for code generation is extremely promising. Such networks can learn patterns in the code, which allows them to generate fragments of software code whose accuracy is much higher than traditional methods. However, according to the study by Kim et al. [39], the use of transformers for code generation is one of the most modern methods, although it has limitations in the context of the interpretability of the results. In this study, hybrid models that combine machine learning and traditional algorithmic methods were proposed for better accuracy and reliability of code generation, especially for complex systems. Hybrid models, contrary to transformational models, can not only generate code but also check its compliance with technical requirements, making them more versatile and suitable for complex applications. According to the study, such models can be used to address the problem of transformers' limitations, providing not only high-quality generation but also integration with other stages of software development.

Test automation is another important area where AI can bring significant optimisation. Traditionally, testing has been time-consuming and resource-intensive, test scenarios and validation of each piece of code for defects are done manually. However, with the development of AI, in particular machine learning algorithms, it has become possible to automatically generate test scenarios, detect defects in the code, and even predict failure points. One approach to test automation uses search algorithms to automatically generate test cases. Although this approach is effective, it requires significant computing resources and does not always provide complete test coverage. More effective are methods that use deep learning to detect defects in the code. Approaches such as the one by Liu et al. [42] can predict defects in the code, but they are limited to certain programming languages or types of applications.

Compared to the results of studies such as by Anik et al. [51], which analysed deep neural networks, traditional methods such as SVM or Naive Bayes demonstrate limited adaptability. For instance, deep neural networks trained on multilingual datasets provide versatility and efficiency in detecting code defects regardless of the programming language, while SVMs and other classical methods often require customisation for specific languages and data types. However, traditional methods have advantages in learning speed and computational efficiency, especially on small datasets.

In addition, active learning approaches used in deep neural networks allow for the automatic identification of critical areas for testing, increasing test coverage, as noted by Anik et al. [51] In this context, the results of the current study demonstrate that classical algorithms, although less versatile, are still an effective choice for narrower tasks, especially when resources or data are limited.

Deep learning can improve the accuracy and efficiency of error detection, as well as automate complex development stages that previously required significant human resources. In the future, the development of such methods can lead to a significant increase in software productivity and quality. Existing research is yielding results, but there are still problems that need to be addressed. It is necessary to develop universal models that can work with different programming languages and types of software. It is also important to test the proposed methods in real software development environments to assess their effectiveness and adaptability to different environments.

Compared to deep neural networks and transformers, hybrid models proposed by Ip [52] integrate machine learning with classical algorithmic approaches. They provide not only generation but also verification of compliance with technical requirements. For example, such models can automatically add documentation to the code and check the style and compliance with standards. However, hybrid models may be less flexible in new environments, as much of their functionality is based on defined rules.

Deep neural networks are beneficial in code generation, but they are inferior to transformers in their ability to adapt to broad contexts. Hybrid models, on the other hand, provide greater reliability and accuracy for specific tasks but are more complex to develop. For real-world software development environments, transformers are preferred due to their versatility, although hybrid models may be better for critical systems with strict requirements.

The study of modern methods of test automation based on the use of search algorithms and neural networks demonstrates significant advantages over traditional approaches such as static testing or manual testing. Such methods can significantly reduce the time to detect errors, increase test coverage, and improve the accuracy of software quality control. However, there are challenges related to the versatility of these methods: they must be able to work effectively in different technological environments and with different programming languages. Compared to traditional approaches, which are limited to

specific environments, new AI-based methods offer more flexibility and scalability, but require additional research to achieve full universality.

Based on the analysis, the best methods for use in software engineering are those that strike a balance between accuracy, adaptability, and efficiency. In code generation, transformers such as Codex perform best for general-purpose tasks, while hybrid models are optimal for specific and critical systems due to their ability to verify compliance with technical requirements. In test automation, methods based on active learning and deep neural networks outperform traditional search algorithms due to their greater test coverage and accuracy, although the latter remains effective for projects with limited resources. In defect detection, deep learning methods offer versatility and high accuracy, especially in multilingual environments, while classical methods such as SVMs are useful for smaller projects due to their speed and simplicity. To increase productivity, it is important to develop models that can be easily integrated into existing development tools (e.g., CI/CD systems) and adapt to the specifics of different environments. The further development of AI methods should be aimed at creating integrated solutions that combine the flexibility of transformers, the accuracy of hybrid models, and the efficiency of traditional algorithms. This will not only improve the quality and productivity of development but also make software engineering more adaptive to the needs of modern technologies.

5 Conclusions

This research evaluated the incorporation of AI into the software development process, emphasising its capacity to streamline processes, decrease development duration, and improve product quality. Critical discoveries indicate that AI applications, especially machine learning and deep neural networks, may markedly enhance several phases of the software life cycle, encompassing requirements analysis, design, testing, and code production. By automating repetitive operations, like code generation and test case production, AI minimises human error, expedites development, and guarantees more stable and dependable software.

The study underscored the efficacy of AI in automating software testing and defect identification, illustrating that machine learning algorithms may identify problems early in the development phase, hence enhancing the overall quality of the product. Furthermore, AI's contribution to natural language processing (NLP) in requirements analysis has demonstrated its utility since NLP models may autonomously convert textual specifications into formal models, hence reducing misinterpretations between developers and clients.

Despite the potential benefits, the research also recognised some problems, such as the necessity for enhanced openness and interpretability of AI models, the ethical ramifications of their use, and apprehensions around data privacy and security. These concerns highlight the necessity of creating responsible AI systems that comply with ethical principles and security protocols to guarantee the dependability and credibility of AI in software development.

Given these limitations, further research should be devoted to the development of universal methods for integrating AI at different stages of software development, including for specific types of programs and applications. It is also necessary to conduct experimental testing of the proposed approaches in real conditions and on various platforms to assess their effectiveness and adaptability to different environments.

References

[1] Edy Susanto and Zahra Dinul Khaq, “Enhancing Customer Service Efficiency in Start-Ups with AI: A Focus on Personalisation and Cost Reduction”, *Journal of Management and Informatics*, 3(2): 267-281, 2024. https://doi.org/10.51903/jmi.v3i2.34

[2] Lianping Chen and Muhammad Ali Babar, “Variability Management in Software Product Lines: An Investigation of Contemporary Industrial Challenges”, in *Proceeding of 14th International Conference: Software Product Lines: Going Beyond*, J. Bosch, J. Lee, Eds., Heidelberg: Springer Berlin, 2010, pp. 166-180.

[3] Burr Settles, *Active Learning*, Cham: Springer, 2012.

[4] Miltiadis Allamanis, Earl T. Barr, Prem Devanbu and Charles A. Sutton, “A Survey of Machine Learning for Big Code and Naturalness”, *ACM Computing Surveys*, 51(4): 81, 2018. https://dl.acm.org/doi/10.1145/3212695

[5] Christoph Molnar, *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*, Munich: Lean Publishing, 2019.

[6] Hadhemi Jebnoun, Md. Saidur Rahman, Foutse Khomh and Biruk Asmare Muse, “Clones in Deep Learning Code: What, Where, and Why?”, *Empirical Software Engineering*, 27: 84, 2022. https://doi.org/10.1007/s10664-021-10099-x

[7] Psico-Smart Editorial Team. “How can Artificial Intelligence be Utilized in Performance Analysis and Evaluation?” PsicoSmart. 2024. [Online.] Available: https://psico-smart.com/en/blogs/blog-how-can-artificial-intelligence-be-utilized-in-performance-analysis-and-evaluation-139540

[8] Paula Boddington, “Towards the Future with AI: Work and Superintelligence”, in *AI Ethics: A Textbook*, Singapore: Springer, 2023, pp. 409-456.

[9] N.M. Anoop Krishnan, Hariprasad Kodamana and Ravinder Bhattoo, “Interpretable Machine Learning”, in *Machine Learning for Materials Discovery: Numerical Recipes and Practical Applications*, Cham: Springer, 2024, pp. 159-171.

[10] Hongjia Li, Tianshu Wei, Ao Ren, Qi Zhu and Yanzhi Wang, “Deep Reinforcement Learning: Framework, Applications, and Embedded Implementations: Invited Paper”, in *2017 IEEE/ACM International Conference on Computer-Aided Design*, Irvine: Institute of

Electrical and Electronics Engineers, 2017, pp. 847-854.

[11] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi and Thomas Zimmermann, "Software Engineering for Machine Learning: A Case Study", in *Proceedings of 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice*, Montreal: Institute of Electrical and Electronics Engineers, 2019, pp. 291-300.

[12] Silvio Andrae, "Cyber Risk Assessment Using Machine Learning Algorithms", in *Advances in Computational Intelligence and Robotics*, Hershey: IGI Global, 2025, pp. 187-218.

[13] Satvik Garg, Pradyumn Pundir, Geetanjali Rathee, P.K. Gupta, Somya Garg and Saransh Ahlawat, "On Continuous Integration / Continuous Delivery for Automated Deployment of Machine Learning Models using MLOps", in J. Gurrola, Ed., *Proceedings of 2021 IEEE Fourth International Conference on Artificial Intelligence and Knowledge Engineering*, Hills: Institute of Electrical and Electronics Engineers, 2021, pp. 25-28.

[14] Samir Jaber, "Intelligent Software Testing and AI-Powered Apps: From Automated Defect Prediction to Context-Aware Mobile Services", 2024. https://doi.org/10.13140/RG.2.2.20694.46401

[15] Milda Maciejauskaitė and Jolanda Miliauskaitė, "The Efficiency of Machine Learning Algorithms in Classifying Non-Functional Requirements", *New Trends in Computer Sciences*, 2(1): 46-5, 2024. https://doi.org/10.3846/ntcs.2024.21574

[16] Mamta Kalra and Suman Sangwan, "Machine Learning and Deep Learning Techniques for Recommendation Systems: A Comprehensive Review", *Journal of Harbin Engineering University*, 45(5): 158-172, 2024.

[17] Vitalii Shymko and Oleg Slipych, "The use of innovative solutions and VR-technologies in architectural design and construction", *Mining Journal of Kryvyi Rih National University*, 58(1): 110-114, 2024. https://doi.org/10.31721/2306-5435-2024-1-112-110-114

[18] Oleksandr Tkachenko, Aleksei Chechet, Maksim Chernykh, Sergei Bunas and Przemysław Jatkiewicz, "Scalable Front-End Architecture: Building for Growth and Sustainability", *Informatica (Slovenia)*, 49(1): 137-150, 2025. https://doi.org/10.31449/inf.v49i1.6304

[19] Seyit Kerimkhulle, Ainur Saliyeva, Ulzhan Makhazhanova, Zhandos Kerimkulov, Alibek Adalbek and Roman Taberkhan, "The estimate of innovative development of construction industry in the Kazakhstan", *E3S Web of Conferences*, 389: 06004, 2023. https://doi.org/10.1051/e3sconf/202338906004

[20] Huaming Chen and M. Ali Babar, "Security for Machine Learning-based Software Systems: A Survey of Threats, Practices and Challenges", *ACM Computing Surveys*, 56(6): 151, 2022. https://doi.org/10.1145/3638531

[21] Itzhak Aviv, Ruti Gafni, Sofia Sherman, Berta Aviv, Asher Sterkin and Etzik Bega, "Infrastructure From Code: The Next Generation of Cloud Lifecycle Automation", *IEEE Software*, 40(1): 42-49, 2023. https://doi.org/10.1109/MS.2022.3209958

[22] Prajakta Sudhir Khade and Dr. Rajeshkumar U. Sambhe, "Artificial Intelligence in Software Development: A Review of Code Generation, Testing, Maintenance and Security", *International Journal of Current Science Research and Review*, 8(4): 1632-1641, 2025. https://doi.org/10.47191/ijcsrr/V8-i4-08

[23] Ruslan Yermolenko, Denys Klekots and Olga Gogota, "Development of an algorithm for detecting commercial unmanned aerial vehicles using machine learning methods", *Machinery and Energetics*, 15(2): 33-45, 2024. https://doi.org/10.31548/machinery/2.2024.33

[24] Sedighe Ajorloo, Amirhossein Jamarani, Mehdi Kashfi, Mostafa Haghi Kashani and Abbas Najafizadeh, "A Systematic Review of Machine Learning Methods in Software Testing", *Applied Soft Computing*, 162: 111805, 2024. https://doi.org/10.1016/j.asoc.2024.111805

[25] Albina Yerzhanova, Seyit Kerimkhulle, Gulzira Abdikerimova, Margaryta Makhanov, Svetlana Beglerova and Zhazira Taszhurekova, "Atmospheric correction of landsat-8 / Oli data using the flaash algorithm: Obtaining information about agricultural crops", *Journal of Theoretical and Applied Information Technology*, 99(13): 3110-3119, 2021.

[26] Oyekunle Claudius Oyeniran, Okechukwu Adewus, Adams Gbolahan Adeleke, Lucy Anthony Akwawa and Chidimma Francisca Azubuko, "AI-Driven Devops: Leveraging Machine Learning for Automated Software Deployment and Maintenance", *Engineering Science & Technology Journal*, 4(6): 728-740, 2023. https://doi.org/10.51594/estj.v4i6.1552

[27] Sabina-Cristiana Necula, Florin Dumitriu and Valerică Greavu-Șerban, "A Systematic Literature Review on Using Natural Language Processing in Software Requirements Engineering", *Electronics*, 13(11): 2055, 2024. https://doi.org/10.3390/electronics13112055

[28] Olga Sokol, "Optimising productivity and automating software development: Innovative memory system approaches in large language models", *Technologies and Engineering*, 26(1): 36-44, 2025. https://doi.org/10.30857/2786-5371.2025.1.3

[29] Suprit Pattanayak, Pranav Murthy and Aditya Mehra, "Integrating AI into DevOps pipelines: Continuous integration, continuous delivery, and automation in infrastructural management: Projections for future", *International Journal of Science and Research Archive*, 13(01), 2244-2256,

2024. https://doi.org/10.30574/ijsra.2024.13.1.1838

[30] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha and Michael P. Wellman, "Sok: Security and Privacy in Machine Learning", in L. O'Conner, Ed., *2018 IEEE European Symposium on Security and Privacy*, London: Institute of Electrical and Electronics Engineers, 2018, pp. 399-414.

[31] Christopher Kuner, Lee A. Bygrave and Christopher Docksey, "Background and Evolution of the EU General Data Protection Regulation (GDPR)", in C. Kuner, L.A. Bygrave, C. Docksey, L. Drechsler, Eds., *EU General Data Protection Regulation* (GDPR), New York: Oxford University Press, 2020, pp. 1-47.

[32] Yundong Zhao, "Optimization of Mechanical Manufacturing Processes Via Deep Reinforcement Learning-Based Scheduling Models", *Informatica*, 49(14): 19-32, 2025. https://doi.org/10.31449/inf.v49i14.7204

[33] Roman Shults, Mykola Bilous, Azhar Ormambekova, Toleuzhan Nurpeissova, Andrii Khailak, Andriy Annenkov and Rustem Akhmetov, "Analysis of Overpass Displacements Due to Subway Construction Land Subsidence Using Machine Learning", *Urban Science*, 7(4): 100, 2023. https://doi.org/10.3390/urbansci7040100

[34] Dmytro Belytskyi, Ruslan Yermolenko, Kostiantyn Petrenko and Olga Gogota, "Application of machine learning and computer vision methods to determine the size of NPP equipment elements in difficult measurement conditions", *Machinery and Energetics*, 14(4): 42-53, 2023. https://doi.org/10.31548/machinery/4.2023.42

[35] Bohdan Cherniavskyi, "Integration of Drones and Dio-Inspired Algorithms into Intelligent Transportation Logistics Systems for Post-war Remediation of Ukraine", *Lecture Notes in Networks and Systems*, 1336: 426-437, 2025. https://doi.org/10.1007/978-3-031-87379-9_39

[36] Vasyl Nesterov, "Integration of artificial intelligence technologies in data engineering: Challenges and prospects in the modern information environment", *Bulletin of Cherkasy State Technological University*, 28(4): 82-92, 2023. https://doi.org/10.62660/2306-4412.4.2023.82-90

[37] Ivan Biliuk, Dmytro Shareyko, Oleg Savchenko, Serhii Havrylov, Andrii Fomenko and Anatolii Tubaltsev, "Machine Calculation of the Problem of Expansion of the Magnetic Field Measurement Grid", in *Proceedings of the 5th International Conference on Modern Electrical and Energy System, MEES 2023*, Kremenchuk: Institute of Electrical and Electronics Engineers, 2023. https://doi.org/10.1109/MEES61502.2023.10402546

[38] Yalamanchili Sangeetha and G. Jaya Lakshmi, "Predicting Software Bugs with Machine Learning Algorithms" in V.L.N. Komanapalli, N. Sivakumaran, S. Hampannavar, Eds., *Select Proceedings of I-CASIC 2020: Advances in Automation, Signal Processing, Instrumentation, and Control*, Singapore: Springer, 2021, pp. 2683-2692. https://doi.org/10.1007/978-981-15-8221-9_251

[39] Seokjun Kim, Jaeeun Jang and Hyeoncheol Kim, "All-In-One: Artificial Association Neural Networks", 2021. https://doi.org/10.48550/ARXIV.2111.00424

[40] Luciano Floridi, Josh Cowls, Monica Beltrametti, Raja Chatila, Patrice Chazerand, Virginia Dignum, Christoph Luetge, Robert Madelin, Ugo Pagallo, Francesca Rossi, Burkhard Schafer, Peggy Valcke and Effy Vayena, "AI4People – An Ethical Framework for a Good AI Society", *Mind and Machine*, 28: 689-707, 2018. https://doi.org/10.1007/s11023-018-9482-5

[41] Reza Shokri and Vitaly Shmatikov, "Privacy-Preserving Deep Learning", in *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, Monticello: Institute of Electrical and Electronics Engineers, 2015, pp. 909-910.

[42] Hengyuan Liu, Zheng Li, Baolong Han, Xiang Chen, Doyle Paul and Yong Liu, "Integrating Neural Mutation into Mutation-Based Fault Localization: A Hybrid Approach", *Journal of Systems and Software*, 211: 112281, 2024. https://doi.org/10.1016/j.jss.2024.112281

[43] Anbarasu Arivoli, "AI for automated bug detection and debugging: a comparative study of current approaches", *International Journal of Business Quantitative Economics and Applied Management Research*, 7(12): 57-66, 2024. https://doi.org/10.5281/zenodo.15236718

[44] Blanco Ruiz, "The Role of Automation in Modern Software Testing", *Journal of Computer Engineering & Information Technology*, 12(4), 2023. https://doi.org/10.4172/2324-9307.1000279

[45] Daiju Ueda, Taichi Kakinuma, Shohei Fujita, Koji Kamagata, Yasutaka Fushimi, Rintaro Ito, Yusuke Matsui, Taiki Nozaki, Takeshi Nakaura, Noriyuki Fujima, Fuminari Tatsugami, Masahiro Yanagawa, Kenji Hirata, Akira Yamada, Takahiro Tsuboyama, Mariko Kawamura, Tomoyuki Fujioka, and Shinji Naganawa, "Fairness of artificial intelligence in healthcare: Review and recommendations", *Japanese Journal of Radiology*, 42, 3-15, 2024. https://doi.org/10.1007/s11604-023-01474-3

[46] Gopinath Kathiresan, "Automated Test Case Generation with AI: A Novel Framework for Improving Software Quality and Coverage", World Journal of Advanced Research and Reviews, 23(02): 2880-2889, 2024. https://doi.org/10.30574/wjarr.2024.23.2.2463

[47] Zoe Kotti, Rfaila Galanopoulou and Diomidis Spinellis, "Machine Learning for Software Engineering: A Tertiary Study", *ACM Computing*

Surveys, 55(12): 256, 2023. https://doi.org/10.1145/3572905

[48] Marco Tulio Ribeiro, Sameer Singh and Carlos Guestrin, ""Why should I Trust you?": Explaining the Predictions of Any Classifier", in: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York: Association for Computing Machinery, 2016, pp. 1135-1144.

[49] Olesia Vasetska, "A study of the electric circuit modelling and simulation software efficiency and their accuracy, speed and ease of use comparison", *Bulletin of Cherkasy State Technological University*, 29(2): 32-44, 2024. https://doi.org/10.62660/bcstu/2.2024.32

[50] Itzhak Aviv, D. Gafni, Sofia Sherman, Bertha Aviv, Asher Sterkin and E. Bega, "Cloud infrastructure from python code–breaking the barriers of cloud deployment", in: *European Conference on Software Architecture, ECSA,* 2023, pp. 1-8. https://www.researchgate.net/profile/Itzhak-Aviv/publication/373897534_Cloud_Infrastructure_from_Python_Code_-breaking_the_Barriers_of_Cloud_Deployment/links/6501edd2808f9268d573dea5/Cloud-Infrastructure-from-Python-Code-breaking-the-Barriers-of-Cloud-Deployment.pdf

[51] Shafayat Mowla Anik, Kevyn Kelso and Byeong Kil Lee, "Efficient Layer Optimizations for Deep Neural Networks", *International Journal of Soft Computing and Engineering,* 14(5): 20-29, 2024. http://www.doi.org/10.35940/ijsce.E3650.14051124

[52] Si Hua Robert Ip, "Research on Explainability of Deep Neural Networks and Its Applications", *Highlights in Science, Engineering and Technology*, 115: 441-450, 2024. https://doi.org/10.54097/ekzm5z29