Context-Aware Code Summarization Using Multimodal Transformer

Sadia Saif
Faculty of Computing, Riphah International University, Lahore, Pakistan
sadiasaif7455@gmail.com

Muhammad Yaseen
Faculty of Computing, Riphah International University, Lahore, Pakistan
muhammad.yaseen@riphah.edu.pk

Umar Farooq Khattak
Faculty of Artificial Intelligence and Frontier Technologies, UNITAR International University Selangor, Malaysia
umar.farooq@unitar.my

Gohar Rahman
Faculty of Computing and Informatics, University Malaysia Sabah (UMS), Kota Kinabalu, Sabah, Malaysia
gohar_315@ums.edu.my (corresponding author)

Received: 29 October 2025 | Revised: 20 November 2025 and 30 November 2025 | Accepted: 3 December 2025

 DOI: https://doi.org/10.48084/etasr.15857

ABSTRACT

Modern software systems continue to grow in complexity, making it increasingly challenging for developers to understand code without clear and up-to-date documentation. This study proposes a multimodal transformer architecture based on CodeT5, enhanced with Abstract Syntax Tree (AST) information to improve both code summarization and the detection of semantic bugs. The suggested framework is designed to capture token-level, structural, and contextual cues, enabling deeper program comprehension than traditional text-only models. The model has been trained and tested on the CoNaLa dataset and compared to baseline and CodeT5 models. Experimental results show substantial improvements, achieving a Bilingual Evaluation Understudy (BLEU) score of 81.34 (an improvement of 44.14 points over CodeT5) and a Recall-Oriented Understudy for Gisting Evaluation–Longest Common Subsequence (ROUGE-L) score of 0.89. These findings confirm that incorporating structural awareness significantly enhances summary relevance and bug-identification capability. The study contributes a scalable, context-sensitive model for automated software understanding and offers strong potential for integration into real-world development tools.

Keywords-code summarization; CodeT5; Abstract Syntax Tree (AST); transformer; software engineering

I. INTRODUCTION

Software systems have become increasingly complex. Software developers frequently encounter poorly documented and unwieldy code. This complexity has necessitated the development of tools that can help simplify code and enhance its maintainability and reliability. Code summarization is one such tool that provides brief descriptions of functions, classes, or code in natural language. Such descriptions reduce the burden on developers and help them quickly understand what a piece of code does. They also conserve time during debugging and facilitate the software documentation process, as noted by authors in [1].

Code summarization has become an inherent component of software engineering and helps developers understand complex code more easily. It automatically generates short natural language descriptions of code functions or classes to improve code maintenance. This assists developers in saving time on manual documentation and allows new developers to understand the logic of a large project within a short time. Additionally, summarization supports teamwork, debugging, and knowledge transfer within a development team. Automated code summarization provides an effective and uniform method of documentation management, particularly as software systems continue to grow in size.

Transformer-based models have been widely used in Natural Language Processing (NLP) applications in recent years and are increasingly being applied to code summarization. As pointed out by authors in [2], transformers outperform earlier models like recurrent neural networks because they can capture long-range relationships and dependencies in sequences. However, source code is naturally structured and not purely sequential, and conventional models tend to ignore this structure, resulting in superficial summaries. To address this limitation, authors in [3] and [4] developed multimodal transformer models, which integrate multiple code representations, including Abstract Syntax Trees (ASTs), Control-Flow Graphs (CFGs), Data-Flow Graphs (DFGs), and raw source code. This integration captures both structural and logical relationships within programs and generates summaries that more accurately reflect program intent and behavior.

Despite these advances, authors in [5] observed that most transformer models remain overly dependent on token-level data and lack structural or contextual understanding, which limits their generalization and precision. This limitation highlights the need for a strategy that incorporates structural, semantic, and contextual information to improve code summarization. In comparison, older rule-based models and recurrent approaches, reviewed by authors in [6] and [7], face issues with scalability and long-sequence learning, whereas newer neural approaches often capture only surface-level semantics. Furthermore, authors in [8] found that deep learning models can automatically extract richer features than manually engineered models.

Existing transformer-based approaches still struggle to capture deeper structural context, reducing their ability to generalize and summarize code with complex logic or nested structures. These observations highlight the need for models that integrate structural, semantic, and contextual information for more reliable code understanding.

The main objective of this study is to develop a multimodal transformer model that combines code text, AST structure, and contextual information to improve summarization accuracy. A secondary objective is to evaluate the model against standard baselines to assess its effectiveness and generalizability. To achieve this, the proposed work develops an improved multimodal transformer model using CodeT5 that incorporates structural, semantic, and contextual inputs, such as ASTs, CFGs, and documentation, to generate context-dependent code summaries. The model will be tested on a benchmark dataset, CoNaLa, to determine its accuracy, robustness, and scalability. This framework is expected to refine automated code summarization by learning syntax, documentation, and program flow. It should assist in code review, support understanding of legacy code, and integrate with development tools, including Integrated Development Environments (IDEs) and testing frameworks, thereby enhancing software reliability through higher-quality code summaries and automated code understanding.

This study contributes by integrating AST-based structural information into CodeT5 to create a multimodal summarization framework. It further provides an empirical evaluation showing substantial improvements over baseline models. The work offers a scalable, context-aware framework that enhances automated code understanding and supports practical software engineering tasks.

Code summarization has evolved alongside advances in deep learning, particularly transformer-based approaches for source code comprehension. Earlier models relied primarily on sequential token representations, whereas recent methods capture both syntactic and semantic features to enhance contextual understanding. This study discusses the most critical literature related to replacing token-based summarization models with more sophisticated multimodal architectures, such as CodeBERT, CodeT5, PLBART, and GraphCodeBERT, which integrate structural and semantic inputs to create richer and more precise summaries. Recent studies from 2024–2025, such as those by authors in [9] and [10], further highlight the growing adoption of multimodal transformers for software comprehension tasks.

A. Context-Aware Code Summarization

Early code summarization models mainly concentrated on token-level and syntactic features, and in most cases, they produced shallow and non-contextual summaries. Transformer-based models have addressed this limitation by including semantic and structural representations. Authors in [9] used Large Language Models (LLMs) on Java code, where contextual information was provided at the method level. This approach improved accuracy but was not evaluated on other programming languages.

Authors in [11] reported that integrating ASTs and positional encoding improved semantic processing but increased redundancy. According to authors in [12], semantic embeddings improved performance but lacked cross-language generalization. According to the study by authors in [10], local and global context enhances the quality of summarization.

Authors in [13] observed that the Bilingual Evaluation Understudy (BLEU) and Recall-Oriented Understudy for Gisting Evaluation (ROUGE) metrics fail to capture summary quality comprehensively, so BLEU and ROUGE should be replaced with a contrastive learning metric closer to human judgment. Authors in [14] emphasized the importance of consistent preprocessing to enable fair comparisons. Authors in [15] incorporated semantic information such as variable names and control-flow into prompts, which enhanced performance at the expense of increased computing time. Authors in [16] introduced BASTS, combining ASTs and control-flow data to provide better summaries, albeit at the cost of expensive hardware.

Overall, the literature indicates that meaningful code summaries require both context and structural information. However, most models remain reliant on single data sources and are not flexible across programming languages. These studies demonstrate that contextual information greatly improves summarization quality, which inspires the integration of structural signals for more precise program understanding.

Although current context-aware summarization models have improved greatly, they largely rely on textual or syntactic features. These features are often not integrated between

different code representations, such as ASTs, CFGs, and execution traces, limiting their ability to fully capture program logic. To address this, authors started to explore hybrid and multimodal structures that consider both structural and semantic aspects of code, enhancing the depth of summarization and the degree of generalization.

B. Multimodal Transformer Models

Recent developments indicate that combining multiple code representations, such as ASTs, CFGs, execution traces, and documentation enables a deeper understanding of code logic and intent. Earlier models, including CodeBERT and GraphCodeBERT, applied natural language transformers to computer programs. CodeBERT was trained to learn paired representations of code and natural language, enhancing the relevance of the large-scale code search and summarization tasks on benchmark datasets, such as CodeSearchNet [17]. GraphCodeBERT included data-flow edges into the attention mechanism, which assisted the model better capture relationships among program variables and statements [18].

Later, encoder–decoder frameworks like PLBART and CodeT5 extended these approaches to support both code understanding and code generation tasks [5]. UniXCoder also enhanced cross-language flexibility by jointly training masked language modeling and contrastive learning objectives, enabling the model to align representations among program languages. Authors in [3] proposed a multimodal transformer that integrates code tokens, control-flow, and data-flow information, which improved BLEU and METEOR scores on CodeSearchNet, but with the cost of complex preprocessing. M2TS, proposed authors in [4], is a multi-scale multimodal transformer that leverages both token and AST embeddings, thus increasing accuracy, but with a low scalability. Authors in [19] proved the increased applicability of multimodal transformers in other fields, such as question answering, which suggests their potential for broader software analysis tasks.

Additional research has explored models that incorporate both syntactic and contextual information. Syntactic and contextual data models such as SynCoBERT and Deep Context Transformer [20], achieved higher accuracy on Defects4J. However, they require significant computational resources, and long input sequences remain challenging. A recent study by authors in [21] emphasizes the need for hybrid deep-learning methods capable of handling diverse input modalities, reinforcing the direction adopted in this study.

Table I summarizes related work, including techniques, datasets, key findings, and limitations.

TABLE I. RELATED WORK SUMMARY

Model / Reference	Technique used	Dataset / domain	Key findings / results	Limitations
M2TS [4]	Multi-scale multimodal transformer using token and AST embeddings	Python and Java code datasets	Increased summarization accuracy via multimodality	Low scalability, complex preprocessing
CodeT5 [5]	Text-to-text transformer for code understanding and generation	Codex GLUE, CodeSearchNet	Excellent contextual summarization	Ignores AST/CFG-based structure
CodeBERT [17]	Transformer pre-trained on code–text pairs	CodeSearchNet dataset	High accuracy in code–text understanding tasks	Limited structural representation
Context-aware code summary generation [9]	LLM with method-level contextual input	Java code repository	Improved summary accuracy with contextual integration	Language-specific, lacks generalization
Structure feature transformer [11]	Transformer with AST-based positional encoding	CodeSearchNet benchmark	Enhanced semantic processing and summary structure	Increased redundancy, overfitting risk
CoCoSUM [12]	Multi-relational Graph Neural Network (GNN) using UML and relational graphs	Java and Python projects	Improved accuracy via structural + semantic fusion	Heavy computation, limited scalability
Programmer visual attention model [10]	Visual attention using local and global context	Java methods dataset	Improved readability and coherence of summaries	Performance drops for long code blocks
Semantic prompt augmentation [15]	Semantic prompting using variable names + control-flow data	Python and Java datasets	Higher accuracy in context-aware summarization	Slower due to heavy computation time
BASTS [16]	AST + CFG-based summarization	Open-source code dataset	Better summaries through structural data	Requires high-end hardware
GraphCodeBERT [18]	Data-flow–integrated attention mechanism	CodeSearchNet, Defects4J	Better variable–statement association	Computationally heavy, limited scalability
Tag-cloud code visualization model [22]	Tag-cloud visualization of source code identifiers	Java object-oriented programs	Provides high-level structural visualization	Not a textual summarization approach
Hybrid deep learning detection model [21]	Deep learning models (CNN, RNN, LSTM) for anomaly detection	Time-series network data	High accuracy, reduced false positives	Domain-specific; not directly for code
Proposed model (this work)	Multimodal transformer with AST + documentation context	CoNaLa dataset	BLEU = 81.34, ROUGE-L = 0.89; strong structural & semantic capture	High computational cost; needs optimization

Overall, multimodal transformer models have high potential, as they combine syntax, structure, and semantics to enhance code summarization and defect forecasting. However, most approaches still face high computational cost, complex preprocessing, limited scalability, or reliance on a single dominant modality, which restricts practical use. These challenges underscore the need for lightweight, structurally aware frameworks, such as the proposed AST-enhanced multimodal model, which aims to balance performance and efficiency.

C. Summary and Research Gap

The literature indicates a definite shift from syntax-based to multimodal, context-aware code understanding. Models such as

CodeBERT, CodeT5, and GraphCodeBERT have improved summarization but still rely on single-modality inputs, such as tokens or ASTs [22, 23]. They do not fully incorporate structural, semantic, and contextual information, limiting their adaptability to large-scale or multi-language systems [18]. Furthermore, reproducibility is constrained by inconsistencies in dataset quality, evaluation metrics, and high computational costs [24].

To address these gaps, this study proposes a multimodal transformer framework that integrates code, documentation, and execution data. This approach aims to improve accuracy, generalization, and scalability of code summarization across a wide range of programming environments.

II. METHODOLOGY

In this section, the research design, dataset, preprocessing, model architecture, training, and evaluation are described as applied in this study. The research aims to develop a multimodal transformer model, which can summarize themes in source code and identify semantic bugs using both text and structural evidence.

A. Research Design

This paper combines deep learning and NLP to analyze program code. The model carries out two key functions: generating short natural-language summaries of code and finding latent logical errors. Contrary to traditional transformer frameworks, which operate with only code tokens, the proposed model incorporates both structural and contextual information, including ASTs and code documentation.

The model accepts both the input code and input meaning and transforms them into token representations. They are processed using a transformer backbone, which extracts semantic and structural features simultaneously. The proposed model will be compared to a baseline text-only transformer (T5-small) to determine the benefit of multimodal features. This experiment aims to demonstrate the improvement in summarization accuracy due to the integration of structural and contextual information.

B. Dataset

The dataset used in this study is CoNaLa (Code/Natural Language) [25], which is a collection of sample Python code with brief natural-language intents stating what the code is doing. It is commonly used in tasks such as code summarization and code understanding. The dataset was cleaned to yield 2,379 valid samples, removing records that lacked either the code snippet or the corresponding intent. CoNaLa was selected because the code blocks are written by real developers, making it suitable for training models that reflect real-world usage. Each record contains an ID, the code snippet, the original intent, and a human-refined rewritten intent. The attributes of the dataset are presented in Table II.

This dataset offers the required tradeoff between code structure and description and is therefore optimal for building code summarization activities. Multi-language datasets such as CodeSearchNet were not used in this study to maintain consistency in language-specific patterns and avoid introducing cross-language variability during model evaluation.

TABLE II. CONALA DATASET ATTRIBUTES

Attribute	Description
question_id	Unique ID for each query
intent	Natural language description of code
rewritten_intent	Human-refined version of intent
snippet	Python code snippet

C. Data Preprocessing

The raw data were preprocessed to eliminate noise and prepare them for training with transformers. Filtering was used first to eliminate unfinished sets of code and intents. Next, text normalization was applied, converting all text to lowercase and removing extra spaces and special characters. Code snippets were also formatted to ensure consistent spacing and indentation. Both code and intents were tokenized with the help of a Byte Pair Encoding (BPE) mechanism that preserves key operators and identifiers. Lastly, the data were separated into 80% training, 10% validation, and 10% testing sets to ensure fair learning and evaluation. This preprocessing produced clean and balanced data suitable for training the model.

D. Model Architecture

The proposed model is built on the CodeT5 transformer, a sequence-to-sequence model trained on large code corpora. CodeT5 is capable of learning both natural language and code syntax, making it suitable for code summarization. Nevertheless, it tends to operate only with one kind of input (tokens). In this study, CodeT5 is expanded into a multimodal tool that supports ASTs as well as documentation, providing richer insight into program logic. A baseline model (T5-small) is used for comparison. It considers the code as a text and ignores structural information, allowing evaluation of whether multimodal inputs enhance performance.

The AST-based module parses each code snippet into a tree, providing the logical organization of the program. AST embeddings are concatenated with code tokens and fed into the transformer encoder. This aids the model to capture loops, conditions, and dependencies that establish program meaning. To detect bugs, a classification head is included in CodeT5 to categorize snippets as buggy or non-buggy. The model shares the same tokenizer and embeddings as the summarization model, but uses binary cross-entropy loss instead of sequence-to-sequence loss making the framework applicable for both code understanding and quality assessment.

In contrast to the baseline T5-small, which processes code as plain text without structural awareness, the proposed multimodal CodeT5 leverages AST-based structural representations, enabling the encoder to learn control flow, hierarchy, and dependency cues, improving summarization accuracy.

E. Training and Evaluation

The models were trained using the AdamW optimizer with adaptive learning rate scheduling. For code summarization, a sequence-to-sequence loss function was employed. Overfitting

was avoided using dropout regularization. Standard metrics were used to evaluate model performance, including BLEU, ROUGE–Longest Common Subsequence (ROUGE-L), Metric for Evaluation of Translation with Explicit Ordering (METEOR), and BERTScore. These metrics were used to measure the similarity between the generated summaries and the reference summaries. Accuracy, F1-score, and Receiver Operating Characteristic – Area Under the Curve (ROC-AUC) were calculated to assess the model's ability to identify logical faults.

F. Experimental Workflow

The overall workflow is illustrated in Figure 1. The process begins with the CoNaLa dataset, which undergoes cleaning, tokenization, and feature extraction. The processed data are then divided into testing and training sets. Two models are trained in parallel: the proposed multimodal CodeT5 (including AST and context) and the baseline T5-small (text-only). After training, both models are evaluated on the test set using the selected metrics to compare performance.

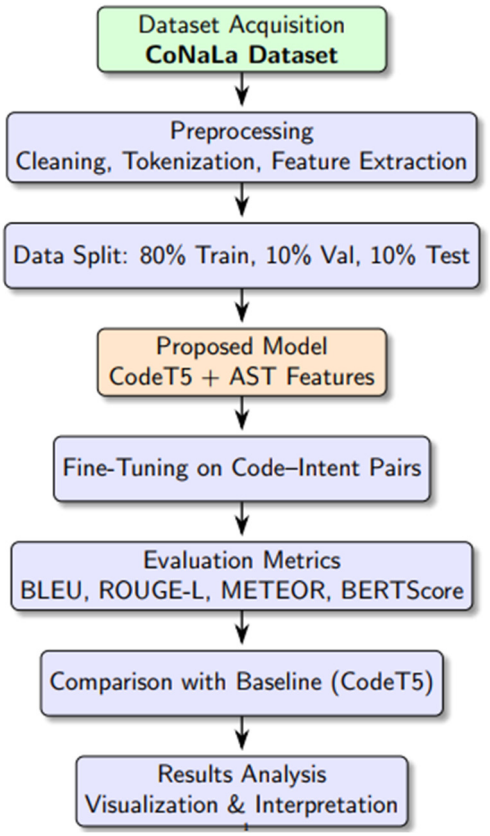

Fig. 1. Workflow diagram of the experimental setup.

III. RESULTS AND DISCUSSION

This section presents the findings of the proposed multimodal transformer model for code summarization using the CoNaLa dataset. The results are compared against two baselines: a text-only transformer (T5-small) and the standard CodeT5 model. The comparison highlights the impact of incorporating structural (AST) and contextual features on summarization quality.

A. Code Summarization Results

The proposed model outperformed both baselines across all evaluation metrics. The baseline transformer produced summaries with limited semantic relevance, whereas CodeT5 demonstrated stronger performance due to its pretraining on large code–text corpora. The AST-augmented model achieved the best results overall, benefiting from its ability to capture structural and hierarchical program relationships.

The summary results are presented in Table III. The BLEU score increased from 6.55 (baseline) to 81.34 (AST model), indicating a substantial improvement in n-gram overlap. Similarly, ROUGE-L and METEOR, which capture sentence structure and semantic alignment, increased significantly, reflecting improved fluency and semantic coherence. The BERTScore reached 0.9814, indicating strong semantic consistency with human-written summaries.

TABLE III. COMPARATIVE CODE SUMMARIZATION PERFORMANCE OF MODELS

Model	BLEU	ROUGE-L	METEOR	BERTScore (F1)
Baseline transformer	6.55	0.3913	0.3624	0.8959
CodeT5	45.20	0.7321	0.7012	0.9427
AST-augmented model	81.34	0.8927	0.8834	0.9814

These results confirm that incorporating AST features enables the model to better capture control flow, function hierarchies, and variable dependencies. The progressive improvement across the baseline, CodeT5, and AST-augmented model demonstrates the cumulative benefits of integrating structural and contextual information.

These improvements are consistent with earlier work by authors in [3] and [4], who also demonstrated that structural representations enhance semantic accuracy and fluency in summarization. The significant gains in BLEU and ROUGE-L further validate the effectiveness of integrating syntactic information into transformer-based models.

The summarization performance across BLEU, ROUGE-L, METEOR, and BERTScore for the baseline transformer, CodeT5, and the proposed AST-augmented model is shown in Figure 2, highlighting the substantial improvements achieved by incorporating structural (AST) information.

B. Discussion

The experimental findings indicate that incorporating structural information significantly enhances the understanding capabilities of transformer-based code models. The AST-augmented model was found to have the best summarization and best bug classification accuracy, which confirms the importance of syntactic and semantic integration to generate results with more meaning and reliability. CodeT5 exhibited high semantic comprehension because of large-scale pretraining, but with the inclusion of AST information it was able to reason about control flow and interdependencies among

variables. Both summarization and defect detection benefited from this enhancement, as the model was able to capture fine-grained program dependencies. Misclassifications were primarily observed in code snippets with limited context or unusual formatting, suggesting that further improvements could be achieved by incorporating more diverse training data and extending the framework to additional modalities, such as CFGs.

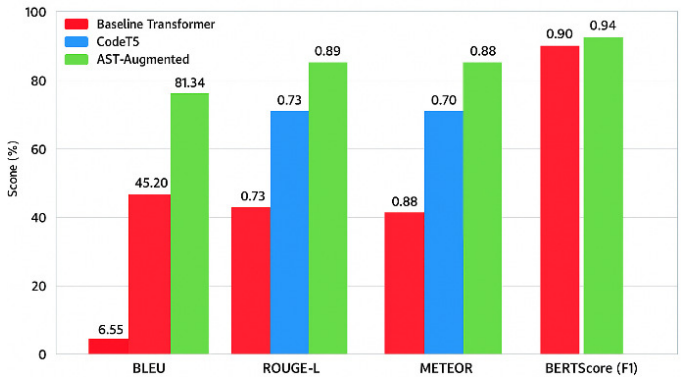

Fig. 2. Comparison of code summarization performance of the models.

IV. CONCLUSION AND FUTURE WORK

This paper presents a multimodal transformer architecture that integrates text and structural information to generate accurate code summaries. Existing transformers, such as CodeT5, primarily rely on token sequences and often lack insight into program structure and logic. The proposed Abstract Syntax Tree (AST)-enhanced CodeT5 model addresses this limitation by leveraging the AST to represent program structure and logical flow.

Evaluation on the CoNaLa dataset demonstrated that the proposed model outperforms both the baseline transformer and standard CodeT5 across all metrics. Specifically, it achieved a 35.7% improvement in Bilingual Evaluation Understudy (BLEU) score over CodeT5, confirming the effectiveness of incorporating AST-based structural signals. The model generated summaries with significantly higher accuracy and stronger semantic relevance, demonstrating that structural awareness is critical for reliable transformer-based code summarization. This framework can assist developers in understanding code more quickly, improving comprehension and productivity, leading to more maintainable and efficient software systems.

Future studies should evaluate the model on larger and multi-language datasets to test its scalability and generalization. Code representations could be further enhanced using Graph Neural Networks (GNNs) or Control-Flow Graphs (CFGs). Additionally, future work should focus on reducing computational costs to enable deployment in real-world applications, such as Integrated Development Environment (IDE) extensions, automated documentation systems, and Continuous Integration/Continuous Deployment (CI/CD) analysis frameworks. These developments will enable more accurate and interpretable Artificial Intelligence (AI)-based software engineering tools.

REFERENCES

[1] A. N. Sontakke, M. Patwardhan, L. Vig, R. K. Medicherla, R. Naik, and G. Shroff, "Code Summarization: Do Transformers Really Understand Code?," presented at the Deep Learning for Code Workshop, Virtual Event, 2022.

[2] S. Gao *et al.*, "Code Structure–Guided Transformer for Source Code Summarization," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 1, Feb. 2023, Art. no. 23, https://doi.org/10.1145/3522674.

[3] Z. Yang *et al.*, "A Multi-Modal Transformer-based Code Summarization Approach for Smart Contracts," in *2021 IEEE/ACM 29th International Conference on Program Comprehension*, Madrid, Spain, 2021, pp. 1–12, https://doi.org/10.1109/ICPC52881.2021.00010.

[4] Y. Gao and C. Lyu, "M2TS: multi-scale multi-modal approach based on transformer for source code summarization," in *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*, Virtual Event, 2022, pp. 24–35, https://doi.org/10.1145/3524610.3527907.

[5] A. Mastropaolo, M. Ciniselli, L. Pascarella, R. Tufano, E. Aghajani, and G. Bavota, "Towards Summarizing Code Snippets Using Pre-Trained Transformers," in *2024 IEEE/ACM 32nd International Conference on Program Comprehension*, Lisbon, Portugal, 2024, pp. 1–12.

[6] X. Zhang, X. Hou, X. Qiao, and W. Song, "A review of automatic source code summarization," *Empirical Software Engineering*, vol. 29, no. 6, Oct. 2024, Art. no. 162, https://doi.org/10.1007/s10664-024-10553-6.

[7] A. Abdu, Z. Zhai, R. Algabri, H. A. Abdo, K. Hamad, and M. A. Al-antari, "Deep Learning-Based Software Defect Prediction via Semantic Key Features of Source Code—Systematic Survey," *Mathematics*, vol. 10, no. 17, Aug. 2022, Art. no. 3120, https://doi.org/10.3390/math10173120.

[8] I. Mehmood *et al.*, "A Novel Approach to Improve Software Defect Prediction Accuracy Using Machine Learning," *IEEE Access*, vol. 11, pp. 63579–63597, 2023, https://doi.org/10.1109/ACCESS.2023.3287326.

[9] C.-Y. Su, A. Bansal, Y. Huang, T. J.-J. Li, and C. McMillan, "Context-aware code summary generation," *Journal of Systems and Software*, vol. 231, Jan. 2026, Art. no. 112580, https://doi.org/10.1016/j.jss.2025.112580.

[10] R. Wallace *et al.*, "Programmer Visual Attention During Context-Aware Code Summarization," *IEEE Transactions on Software Engineering*, vol. 51, no. 5, pp. 1524–1537, May 2025, https://doi.org/10.1109/TSE.2025.3554990.

[11] K. Yang *et al.*, "An Extensive Study of the Structure Features in Transformer-based Code Semantic Summarization," in *2023 IEEE/ACM 31st International Conference on Program Comprehension*, Melbourne, Australia, 2023, pp. 89–100, https://doi.org/10.1109/ICPC58990.2023.00024.

[12] S. Kotsiantis, V. Verykios, and M. Tzagarakis, "AI-Assisted Programming Tasks Using Code Embeddings and Transformers," *Electronics*, vol. 13, no. 4, Feb. 2024, Art. no. 767, https://doi.org/10.3390/electronics13040767.

[13] A. Mastropaolo, M. Ciniselli, M. Di Penta, and G. Bavota, "Evaluating Code Summarization Techniques: A New Metric and an Empirical Characterization," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, Lisbon, Portugal, 2024, pp. 1–13, https://doi.org/10.1145/3597503.3639174.

[14] E. Shi *et al.*, "On the evaluation of neural code summarization," in *Proceedings of the 44th International Conference on Software Engineering*, Pittsburgh, PA, USA, 2022, pp. 1597–1608, https://doi.org/10.1145/3510003.3510060.

[15] T. Ahmed, K. S. Pai, P. Devanbu, and E. Barr, "Automatic Semantic Augmentation of Language Model Prompts (for Code Summarization)," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, Lisbon, Portugal, 2024, pp. 1–13, https://doi.org/10.1145/3597503.3639183.

[16] C. Lin, Z. Ouyang, J. Zhuang, J. Chen, H. Li, and R. Wu, "Improving Code Summarization with Block-wise Abstract Syntax Tree Splitting,"

in *2021 IEEE/ACM 29th International Conference on Program Comprehension*, Madrid, Spain, 2021, pp. 184–195, https://doi.org/10.1109/ICPC52881.2021.00026.

[17] Z. Feng *et al.*, "CodeBERT: A Pre-Trained Model for Programming and Natural Languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, Virtual Event, 2020, pp. 1536–1547, https://doi.org/10.18653/v1/2020.findings-emnlp.139.

[18] D. Guo *et al.*, "GraphCodeBERT: Pre-training Code Representations with Data Flow," in *9th International Conference on Learning Representations*, Virtual Event, 2021, https://doi.org/10.48550/arXiv.2009.08366.

[19] Y. Huang *et al.*, "Context-aware Bug Reproduction for Mobile Apps," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, Melbourne, Australia, 2023, pp. 2336–2348, https://doi.org/10.1109/ICSE48619.2023.00196.

[20] X. Wang *et al.*, "SynCoBERT: Syntax-Guided Multi-Modal Contrastive Pre-Training for Code Representation." arXiv, Sept. 09, 2021, https://doi.org/10.48550/arXiv.2108.04556.

[21] A. A. A. Mohammed, "Improving Intrusion Detection Systems by Using Deep Learning Methods on Time Series Data," *Engineering, Technology & Applied Science Research*, vol. 15, no. 1, pp. 19267–19272, Feb. 2025, https://doi.org/10.48084/etasr.9417.

[22] T. M. Rocha and A. L. D. C. Carvalho, "SiameseQAT: A Semantic Context-Based Duplicate Bug Report Detection Using Replicated Cluster Information," *IEEE Access*, vol. 9, pp. 44610–44630, 2021, https://doi.org/10.1109/ACCESS.2021.3066283.

[23] A. Bansal, "Context-Aware Models for Automatic Source Code Summarization," Ph.D. dissertation, Division of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN, USA, 2024.

[24] D. Roy, S. Fakhoury, and V. Arnaoudova, "Reassessing automatic evaluation metrics for code summarization tasks," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Athens, Greece, 2021, pp. 1105–1116, https://doi.org/10.1145/3468264.3468588.

[25] P. Yin, B. Deng, E. Chen, B. Vasilescu, and G. Neubig, "Learning to mine aligned code and natural language pairs from stack overflow," in *Proceedings of the 15th International Conference on Mining Software Repositories*, Gothenburg, Sweden, 2018, pp. 476–486, https://doi.org/10.1145/3196398.3196408.