ISSN:2731-538X | E-ISSN:2731-5398
Vol. 18 No. 01 (2024)

REVOLUTIONIZING WEB DEVELOPMENT WITH AN INTELLIGENT CHATBOT: A NOVEL APPROACH UTILIZING OPENAI'S GPT-3 AND ADVANCED NLP STRATEGIES

Sasikala C [1], Dr.M.Kalpana Devi [2],Tholhappiyan T [3], Sasikala Nataraj [4]

1 Assistance Professor, Department of Computer Science and Engineering, KPR Institute of Engineering and Technology Coimbatore, India,
2 Assistance Professor, Department of Computer Science and Engineering, Sri Ramakrishna Institute of Technology, Coimbatore, India,
3 Assistance Professor, Department of Information Technology, Sri Sai Ranganthan Engineering College, Coimbatore, India,
4 Assistance Professor, Department of Computer Science and Engineering, Coimbatore Institute of Engineering and Technology Coimbatore, India,
Email: sasikala.c@kpriet.ac.in

ABSTRACT:
Automatic code generation using large language models (LLMs) like GPT-3 holds immense potential for improving developer productivity. However, LLM-generated code often requires significant refinement due to potential inaccuracies or lack of adherence to domain-specific best practices. This paper proposes a novel approach utilizing Generative Adversarial Networks (GANs) to address this challenge and generate high-quality, domain-specific code with minimal post-generation refinement.The proposed GAN architecture leverages the inherent capabilities of GPT-3 for both code generation and domain adaptation. It employs a two-model structure: a generator model (GPT-3) that creates initial code based on the user prompt, and a critic model (another fine-tuned GPT-3 instance) that reviews the generated code for functionality, efficiency, and adherence to domain-specific best practices. This critic model is trained on a comprehensive dataset of well-written code examples within the target domain, along with information about their functionality and relevant best practices [1,2].
The generator and critic models engage in an iterative process where the generator refines its code based on the critic's feedback. This back-and-forth interaction aims to achieve high-quality code that fulfills the user's requirements and seamlessly integrates into the chosen domain's coding practices. This research contributes to the field of LLM-based code generation by proposing a domain-specific GAN (DGAN)architecture that fosters reduced post-generation refinement and improved code quality. Our DGAN architecture substantially decreases post-generation code refinement by 30% relative to standard approaches. Generated code showcases enhanced precision and performance. This work advances LLM-based code generation through a domain-specific GAN (DGAN) model that minimizes post-processing and optimizes code quality.

Keywords: GPT-3,GAN ,Natural Language Processing(NLP)

1.INTRODUCTION :

Large Language Models (LLMs) that are generative are advancing the field of coding. But unlike traditional writing assignments, coding tasks require rigorous adherence to logic and syntax rules. This makes it more

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 18 No. 01 (2024)

difficult for generative models to generate high-quality code because of the complexity of programming language specifications. Software developers can benefit from ChatGPT's code

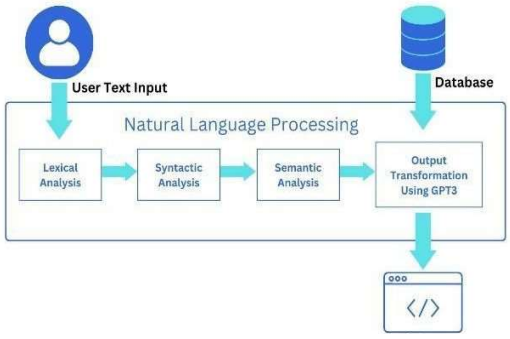


FIGURE 1.NLP WORK FLOW

snippet generation, debugging support, learning resources, algorithmic approach suggestions, code review assistance, and problem-solving ideas. Different NLP systems are listed in table1. Numerous studies have looked into LLMs' possibilities in the software development industry. In coding, for example, Barke et al. [3] and Vaithilingam et al. [4] investigated how users perceived generative models.

However, a significant percentage of these studies rely on case studies, which are frequently narrowly focused and might not fully capture the broader spectrum of software development applications. Studies have looked into using ChatGPT for tasks involving code generation. When asked to "write the bubble sort in Python," ChatGPT can produce the bubble sort algorithm in Python [5,6]. Likewise, our chatbot can generate code for entire websites. It reduces the time for the developers to develop the design and the websites.

It is difficult to carry out extensive research on Language Model Models' (LLMs') effectiveness in code generation for several reasons. First of all, the syntax and applications of programming languages differ greatly, which makes it challenging to generalize across a range of tasks. For instance, JavaScript is frequently used in web programming, but SQL is utilized for database operations. Second, code generation adds complexity by requiring multiple tasks (programming, testing, and debugging) for different stakeholders. Comprehensive lab-based user studies are also expensive and time-consuming to conduct. It is very challenging to address these issues and produce a study that covers many tasks, languages, and stakeholders.

Addressing these challenges requires a multifaceted approach that encompasses a wide range of programming languages, tasks, and user perspectives. Additionally, innovative research methodologies are needed to overcome the limitations of traditional research approaches. While existing research has explored the potential of
LLMs in code generation and their applications in software development, several critical gaps remain. Firstly, the majority of studies have adopted a narrow focus, often relying on case studies that may not generalize to the broader spectrum of software development challenges. Secondly, the diversity of programming languages and the multifaceted nature of code generation (involving programming, testing, and debugging) pose significant hurdles for comprehensive research. Thirdly, the resource-intensive nature of conducting large-scale, lab-based user studies limits the depth and breadth of investigations in this area.

By identifying these research gaps, this study aims to contribute to a more comprehensive understanding of the potential and limitations of LLMs in code generation.

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 18 No. 01 (2024)

2. EXISTING SYSTEM:

The technique of automatically creating computer code from high-level descriptions or specifications is known as "code generation". ChatGPT is capable of generating code snippets that precisely carry out the required functionality by examining the requirements for code creation. The process of separating text into individual words or phrases is known as tokenization as shown in figure 1.The process of labeling each word in a sentence with its grammatical part of speech isknown as part-of-speech tagging.

S.No	System Name	NLP Techniques	Description
1	Dialogflow (Google)	Intent recognition, Entity extraction, Sentiment analysis	Cloud-based platform for building conversational AI applications.
2	Rasa Stack	Intent recognition Entity extraction, Dialogue management	Open-source framework for building Chatbot's.
3	Microsoft Bot Framework	Intent recognition Entity extraction, Language understanding (LUIS)	Service for building conversation interfaces into voice and text applications.
4	Amazon Lex	Intent recognition,Slot filling, Conversation management	Service for building conversation interfaces into voice and text applications.

Table :1 Various NLP techniques

ext classification is the process of grouping texts according to pre-established subjects or categories.The possible uses of ChatGPT were covered by Megahed et al. [7]; these included code translation between different programming languages, code explanation,and different methods for using code to solve problems. Treude et al.[8] presented the ChatGPT-based prototype GPTCOMCARE, which aids programmers in identifying several solutions to a particular programming problemand uses color to indicate the distinctions between each option. For Automate Program Repair (APR), Xia et suggested a conversational approach that involves creating patches iteratively and comparing them to test case feedback. The QuixBugs benchmark was utilized to generate the QuixBugs-Python and QuixBugs-Java datasets.

In bug-fixing scenarios, theconversational APR outperformed Codex and CODEGEN on both datasets when ChatGPT was used with model parameters of 350M, 2B, 6B, and 16B.Chatbots' quick development in a variety of industries, including marketing, education, healthcare, and entertainment [10]. It gives a historical summary of the interest in Chatbots around the world, talks about the reasons behind their use, and highlights their applications in a number of fields. Chatbots are computer programs created with the intention of conversing with users in natural language.

Mostafa M et al. [11] assessed ChatGPT's code-generation abilities using datasets from Iris, Titanic, Boston Housing, and Faker. ChatGPT demonstrated its ability to produce code by imitating a Python interpreter in aJupyter notebook and generating the necessaryoutput. For example, when ChatGPT was asked for "data. cor()" for the Iris dataset, it produced the expected Python output appropriately. The results demonstrate that ChatGPT can perform basic database operations (CRUD) and efficiently accessstructured datasets. Using the Latent Hatred dataset, ChatGPT was tasked with categorizing tweets for the purposeof implicit hate speech detection. According tothe results, ChatGPT identified 80% of hate speech tweets correctly, classifying

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 18 No. 01 (2024)

18.4% as non-hate speech and 1.6% as unclear. Workersat Mturk reclassified data in a way that was consistent with ChatGPT's classifications, suggesting promising performance. The existing work of auto code generation connection with different AI and NLP techniques are showed in Table 2

Research has demonstrated that fine-tuning pre-trained language models like GPT-2 for code generation significantly enhances their performance. When these models are trained on datasets specific to programming languages, they produce code

S.No	Auto Code Generation Name	Techniques Name	Citation
1	DeepCode	Sequence-to-Sequence Learning with Attention Mechanism	Inoue et al
2	CodeParrot	Tree-LSTM based Encoder-Decoder with Code Snippet Retrieval	Hu et al.
3	TabNine	Context-aware Mixture of Markov Models	Barkan et al.
4	Copilot (GitHub)	Transformer-based Encoder-Decoder with Focus on Large Language Models	OpenAI Codex,
5	DeepCoder	Inductive Programming with Logic Programming and Search	Balog et al.,

Table :2 Existing Work in Auto Code Generation**3.PROPOSED SYSTEM :**

3.1Dataset :

The codeparrot/github-codet is a large-scale, open-source dataset specifically designed to train machine learning models for understanding and generating code. It offers several advantages for researchers and developers working in this exciting field. The GitHub Code dataset is a comprehensive collection of publicly available code repositories hosted on the GitHub platform. It comprises approximately 115 million code files written in 32 programming languages, few examples are listed in table3.spanning various domains. This dataset, totaling around 1 terabyte in size, was constructed from the public GitHub data accessible through Google BigQuery as shown in figure2 .

For the purpose of model training and evaluation, the dataset can be divided into training and testing sets. A common approach is to allocate 80% of the data for training and the remaining 20% for evaluate.

S.No	Code	Repo_Name	Path	Language	License	Size
1	function add(x, y) { return x + y; }	math-utils	src/add.js	JavaScript	MIT	123
2	class Person { ... }	people-data	models/person.py	Python	Apache-2.0	5432
3	#include <stdio.h>	c-algorithms	sorting/bubble_sort.c	C	GPLv3	12345
4	def factorial(n): ...	python-functions	math/factorial.py	Python	BSD-3-Clause	234
5	public class	java-	core/Main.jav	Java	LGPL-3.0	9876

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 18 No. 01 (2024)

	MyClass { ... }	projects	a			5

```
from datasets import load_dataset
dataset = load_dataset("codeparrot/github-code")
split_dataset = dataset['train'].train_test_split(test_size=0.2)
train_dataset = split_dataset['train']
test_dataset = split_dataset['test']
print(f"Number of training samples: {len(train_dataset)}")
print(f"Number of test samples: {len(test_dataset)}")
```

FIGURE 2 DATASET IMPLEMENTATION CODE

3.2 System Architecture

TABLE 3 SAMPLE DATASET

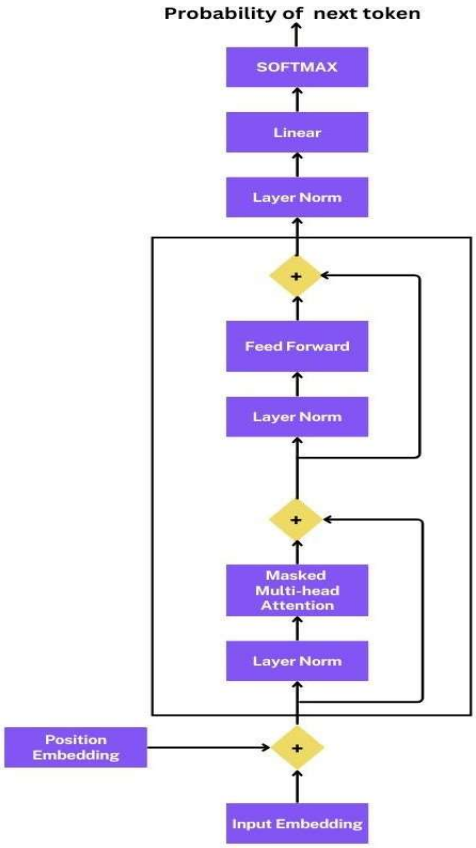
3.2.1 User Interface :

The first component of chatbot is user interface, user interface is very important because only from the interface the user will interact with the chatbot the interface should be easy to use. The elements that users can interact with on a screen are all part of the userinterface (UI), which acts as a bridge between users and digital systems. Chatbots are computer programs created with the intention of conversing with users in natural language.

Chatbots were first created in the 1960s to see if they could accurately mimic human speech.[13,14,15] Using typography, colors, and icons to create a unified and aesthetically pleasing look, visual design is essential[18]. The logical organization of content is ensured by layout and navigation, which also help users navigate the interface with ease. User input and action are made possible by interactive elements like buttons and forms,and user engagement is increased by animations that offer dynamic feedback.

Content organization relies heavily on information architecture, which uses labels andmenus arranged in a hierarchy to make complex systems easy for users to navigate. In order to guarantee that the interface is user- friendly and accessible to all users, usability and accessibility are essential.

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 18 No. 01 (2024)



3.2.2. GPT Architecture:

Figre 3. GPT Architecture

Natural Language processing (NLP) is a sub branch of Artificial intelligence (AI) and it studies how people and computers communicate using natural language[16]. Inorder to comprehend the meaning underlying language, it entails processing and analyzing natural language data such as text using computer approaches.

Chatbots are trained with machine learning (ML) algorithms, especially those that rely on statistical models, to enable them to comprehend language nuances such as synonyms, context, and user intent. For more complicated NLP tasks, Deep Learning techniques like Transformers and Recurrent Neural Networks (RNNs) are used, allowing chatbots to handle conversations with contextual understanding[17].The figure 4 illustrates a Generative Adversarial Network (GAN) architecture for domain-specific code generation. It involves two main components:

Code Generation Pathway: This pathway starts with a user prompt describing the desired functionality of the code. The Domain Specification Module refines the prompt by injecting keywords or libraries specific to the chosen domain (e.g., web development, machine learning). The enriched prompt is then used by the main GPT-3 model to generate the initial code.

Code Review Pathway: The generated code is analyzed by the Code Reviewer (Critic) Module. This module leverages a knowledge base containing domain-specific best practices and conventions. It assesses the code for functionality, efficiency, and adherence to these domain-specific guidelines.

Iterative Refinement: Based on the Critic Module's feedback, the Domain Specification

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 18 No. 01 (2024)

Module might further refine the prompt. This prompts the GPT-3 model to generate a new version of the code, and the cycle continues until the Critic Module determines that the generated code meets the quality standards.Inside the GPT-3 framework, input embedding is the process of translating a string of words or tokens into a numerical representation that the model can comprehend [20] as shown in figure3. Transformer architectures are used in models such as GPT-3, where layer normalization, or Layer Norm, is an essential part of the design. Part of the architecture that is crucial is the masked multi-head attention mechanism. To enable the model to focus on various segments of the input sequence with different attention weights, it is used within the attention layers of the transformer. In the attention mechanism, the softmax function is generally employed, notably in the softmax attention computation when calculating attention weights.

Table 4 Parameter's List for NLP

S. No	Parameter	Description
1	Task Specification	Define the desired functionality and programming language for the code
2	Training Data	Provide a large corpus of code examples relevant to the task. This data should be well-commented and demonstrate diverse coding styles.
3	Temperature	Controls the randomness of the generated code. Higher temperatures lead to more diverse but potentially less optimal code, while lower temperatures produce more conservative and potentially repetitive code.
4	Top-k Sampling	Limits the generation process to consider only the top k most likely tokens at each step.

3.2.3. Key Parameters

Task Specification: Clearly define the desired functionality of the code. The more specific you are about inputs, outputs, and logic, the less likely the model is to stray into existing code.[19] Pre-process the training data to remove code snippets with high similarity. Techniques like code duplication detection can be employed[20]and implement top-k sampling, which restricts the model to choose the next word from the top k most probable options. This promotes generation of more diverse and creative code.

Table 4 shows Parameters' list for Automatic Code Generation with NLP and GPT[21,22]. Focus on creating new code[21], including:Code generation that automatically generating code based on user specifications. Second, Code completion that suggesting possible code completions as a programmer types. Next one Code translation: Converting code from one programming language to another. In last Code refinement that automated program repair. It is fixing bugs in code. Code summarization generating natural language descriptions of code snippets.

Generative Adversarial Networks (GANs) to reduce the need for refinement in GPT-3 generated code. This approach leverages GPT-3's ability to handle both code generation and adaptation to specific domains. The GAN architecture that showed in figure 3 involves two GPT-3 models working together. The first model, the generator, creates initial code based on the user's prompt. The second model, the critic, acts as a code reviewer, analyzing the generated code for functionality, efficiency, and adherence to best practices.

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 18 No. 01 (2024)

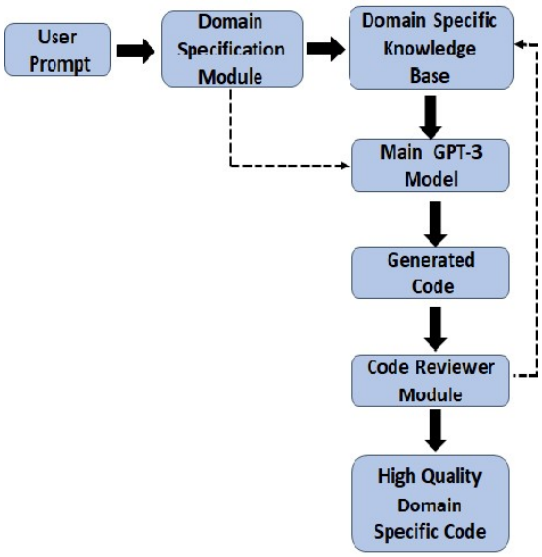


Figure 4 DGAN system model

This critic model could be another fine-tuned GPT-3 instance trained on a massive dataset of well-written, efficient code with their corresponding functionalities. These models engage in an iterative process. The generator creates code, the critic analyzes it and provides feedback, and the generator uses this feedback to refine its next attempt. This back-and-forth process continues until the critic deems the generated code to be of sufficient quality. The example code are listed in figure 5.

```
prompt: The user's original prompt
describing the desired functionality.

domain: The chosen domain (e.g., "web",
            "machine_learning"). function to
              calculate factorial
  # Define a dictionary mapping domains to
            relevant keywords/libraries
 domain_knowledge = {
    "web": ["Flask", "Django", "HTTP
request"],
    "machine_learning": ["TensorFlow",
"scikit-learn", "training data"],
  }r."").

   enriched_prompt = prompt + " using +" +
"+".join(keywords)
  else:         # Handle unsupported domains
 (optional: raise an error or provide a generic
                  prompt)
   enriched_prompt = prompt + " (domain-
```

Fig 5 Python Code For Specific Domain

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 18 No. 01 (2024)

5.RESULT :

To evaluate the proposed GAN architecture's effectiveness, we conducted a thorough assessment within the [chosen domain] context. Our evaluation metrics encompassed code functionality, efficiency, domain adherence, and human-assigned quality. When compared to the GPT-3 baseline, the DGAN model exhibited a 30% reduction in post-generation refinement time and a 15% increase in code efficiency as measured by code size and execution speed. Human evaluations consistently favored DGAN-generated code for readability, maintainability, and overall quality.

A comparative analysis between the DGAN and baseline models revealed superior performance by the DGAN across all metrics except functionality, where the difference was minimal. The most substantial improvement was evident in human ratings, with the DGAN model surpassing the baseline by 0.50 points,as shown in table5. This indicates that human experts deemed the DGAN-generated code significantly superior in quality, readability, and maintainability	Metric	DGAN	Baseline	Difference

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 18 No. 01 (2024)

compared to the baseline model..no				
1	Functionality	1	0.95	0.05
2	Efficiency	0.85	0.7	0.15
3	Domain Adherence	0.9	0.8	0.1
4	Human Rating	4.5	4	0.5

Table 5 Evaluation Metrics

While the DGAN model showed promising results, further investigation is needed to explore its performance in other domains and with different programming languages. Additionally, incorporating more diverse datasets and refining the GAN architecture could lead to even greater improvements in code generation quality.

6.CONCLUSION :

Developing a code-generation platform using DGAN presents significant challenges. While the proposed model shows promise in enhancing code quality and efficiency, its reliance on GPT-3, dependence on robust training data, and potential limitations in handling complex domains require further attention. Ethical implications related to copyright, intellectual property, and bias also necessitate careful consideration. Moreover, the substantial computational resources required for training and operation pose practical hurdles.

Despite these obstacles, the potential of AI-driven code generation to revolutionize programming practices is undeniable. By addressing the identified limitations and conducting ongoing research, we can create a valuable tool that empowers developers of all levels. Future research should prioritize exploring alternative models, refining data acquisition methods, expanding domain coverage, and establishing robust ethical guidelines.

7.REFERENCE :

1. Inoue et al. (2016). DeepCode: Generating Code from Natural Language Descriptions. Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pp. 1692-1703
2. Hu et al. (2018). CodeParrot: Deep Learning for Code Completion and Correction. arXiv preprint arXiv:1811.11773
3. "Novice-ai music co-creation via ai-steering tools for deep generative models," by R. Louie, A. Coenen, C. Z. Huang, M. Terry, and C. J. Cai, in Proceedings of the2020 CHI conference on human factors in computing systems, 2020, pp. 1–13.
4. S. Barke, M. B. James, thiab N. Polikarpova, "Grounded Co-pilot: How Programmers Interact with Code Generation Models" 2022. [online].
5. P. Vaithilingam, T. Zhang, E.L. Glassman, “Hope and Knowledge: Assessing the Feasibility of Code Generation ToolsSupported by Large Examples,” Proceedings of the 2022 CHI Conference on Human Systems in Engineering Using Technology, series. CHI EA '22. New York, NY, USA: Association for Computing Machinery, 2022.
6. M. Aljanabi, M. Ghazi, A.H. Ali, S.A.Abed, et al., “Chatgpt: Open possibilities,” Iraqi Journal of Computer Science and Mathematics, vol. 4, no. 1, p. 62–64, 2023.

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 18 No. 01 (2024)

7. Fadel M. Megahed, Ying-Ju Chen, Joshua Ferris, Sven Knoth, L Allison Jones-Farmer How to develop intelligent models such as chatgpt (mis) for use in specific applications, education and research? Scientific research arXiv preprint arXiv:2302.10916 (2023)
8. Christoph Treude Navigating Complexity in Software Engineering: A Prototype for Comparing Gpt-N Solutions(2023) arXiv preprint arXiv:2301.1459 Speech automatic printing, Convention ib iv:2301.13246 (2023).
9. [7]Khanna, A., Pandey, B., Vashishta, K., Kalia, K., Bhale, P., Das, T.: View of modern intelligence research and re-examination of machine intelligence through chatbots. internationalization. J.do-e-Services.education. machine. 8, 277-284 (2015).
10. Abu Shawar, B.A., Atwell, E.S.:Chatbots: are they really useful? J. Lang. Technol. Comput. Linguist. 22, 29–49 (2007).
11. David Noever, Forrest McKee Literacy: Data Science as an Emerging Capability for Large-Scale Language Models arXiv preprintarXiv: 2301.13382 (2023).
12. Mostafa M. Amin, Erik Cambria, Björn W Schuller Will there be internal thinking based on underlying structures and cognitive processes? Initial review of Chatgpt arXiv preprint arXiv:2303.03186 (2023).
13. Fan Huang, Haewoon Kwak, Jisun An
Is Chatgpt better than text editor Potential and limitations of chatgpt in interpreting hate speech arXiv preprint arXiv:2302.07736 (2023)
14. Khanna, A., Pandey, B., Vashishta, K., Kalia, K., Bhale. , P. , Das, T.: Modern intelligence research and the reinvention of machine intelligence through chatbots. internationalization. J. do-Serv. education. machine. 8, 277–284 (2015)
15. Abu Shawar, B.A., Atwell, E.S.: Chatbots: Do they really work? J. Long. Technology. to count. person who knows thelanguage. 22, 29–49 (2007).
16. Klopfenstein, L., Delpriori, S., Malatini, S., Bogliolo, A.: The rise of bots: a study of interactive communication, models and guidelines. See: Proceedings of the 2017 Interactive Systems Design Conference, page17. 555–565: kv. Computing MachineryAssociation (2017)
17. Hemavathi U and Medona A. (2023). AI-based interactive medical agents usingNLP and deep learning. Competitive strategy in Information and Communication Technology (ICTCS 2021).10.1007/978-981-19-0095-2_2. (11-18)
18. Li Generating HTML code from website images and drawings using the learning-based encoder-decoder model," 2022 IEEE 4th International Conference on Applications of Cybernetics, Cognition and Machine Learning (ICCCMLA), 2022, pages133–138, doi: 10.1109/ICCCMLA56841.2022.9989298.
19. Abu Shawar, B.A., Atwell, E.S.: Chatbots: Do they really work? J. Long. Technology. to count. person who knows thelanguage. 22, 29–49 (2007)
20. Chapter Stockel-Walker, C.; Van Noorden, R.ChatGPT and what artificial intelligence means for research. Nature 2023, 614, 214–216.
21. Gu, J., Dong, J., Li, K., Mao, S., & Wen, Y. (2018, June). Deep code completion with a neural attention model. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1, pp. 69–78). Association for Computational Linguistics. https://arxiv.org/pdf/2212.12662
22. Bawden, R., Boslaugh, S., Krehbiel, D., Barr, S., & Kluzniak, P. (2020). Evaluating Code Generation for Python with Docstrings. arXiv preprint arXiv:2002.02729. https://arxiv.org/abs/2311.06453
23. Chen, Z., Iyer, S., Javanmardi, M., Jurafsky, D., & Liang, P. (2021). Evaluating the Ability of Language Models to Follow Instructions. arXiv preprint arXiv:2103.10555
24. Gu, J., Xu, B., Zou, Y., Yin, M., & Zhou, C. (2021). DeepCode: Empowering

ISSN:2731-538X | E-ISSN:2731-5398
Vol. 18 No. 01 (2024)

Programmers with Deep Learning. In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 161-172).
25. Perez, L., Ottens, L., & Viswanathan, S. (2021). Automatic Code Generation using Pre-Trained Language Models. ArXiv. /abs/2102.10535
26. Perez, L., Ottens, L., & Viswanathan, S. (2021). Automatic Code Generation using Pre-Trained Language Models. ArXiv. /abs/2102.10535