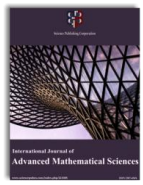
International Journal of Advanced Mathematical Sciences, 11 (1) (2025) 49-54

International Journal of Advanced Mathematical Sciences

Website: www.sciencepubco.com/index.php/IJAMS
https://doi.org/10.14419/s00a4t09
Research paper



Codex comment: lexical and syntactic enhanced code comment generation

Pankaj Ganage [1] *, Shrutika Mahajan [1], Sachin Jadhav [1], Samarth Gupta [1], Mrs. Jayshri Dhere [1, 2], Ms. Abhilasha Bhagat [1, 2]

[1] Dr. D. Y. Patil Institute of Engineering, Management and Research, Akurdi, Pune - 44, India
[2] Dr. D. Y. Patil International University, Akurdi, Pune - 44, India
**Corresponding author E-mail: pankajganage22@gmail.com*

Received: February 26, 2025, Accepted: March 13, 2025, Published: March 20, 2025

Abstract

Insufficient or poor-quality code comments are the result of inexperience or intentional omission, which can hinder software comprehension and pose challenges, particularly in the context of maintenance within a team-programming environment. Here, the smart tools for a more detailed understanding, such as automatic generation of comprehensible code comments, can give insights about functionality and purpose of every element of the code. During the last few years, studies were conducted about this problem since code comments are the only way for human constructively to code is to be conveyed and are therefore highly important to developers. The most popular solutions are inspired by recent breakthroughs in NMT in casting the problem of generating code comments as an NMT problem consisting of translating code into sounding-the-best-of-human comments. With such a level of advancement, we will be introducing Codex Comment, the latest and more advanced deep learning-based approach, to automatically generate high-quality comments for any code. The following is based on the Transformer model first proposed by Google researchers, which achieved impressive results in solving diverse sequence-to-sequence tasks to achieve faithful code comment generation. Codex Comment generates hybrid codes, which carry both lexical as well as syntactic information, hence a more structure-sensitive representation of source code compared to others in sub-word-based approaches. As in the codex model, it generates the harmonic information from its tokens of operations.

Keywords: *Program Comprehension; Code Comment Generation; Hybrid Code Representation; Transformer Model; Sim SBT; AST.*

1. Introduction

Although code comment writing is the most vital source of program comprehension, most developers approach the task with experience. In such a case, what happens is poor code documentation that eventually becomes too complicated to maintain. These are the problems automated tools producing exact code comments have become so popular. This is an innovative approach using a deep transformer model with a hybrid code representation, encompassing lexical and syntactic knowledge. This will indeed handle the out-of-vocabulary tokens through Byte-BPE and effectively traverse the SBT-style ASTs, potentially yielding insightful context-aware comments to improve code readability as developer productivity.

1.1. The importance of code comments

Comments in the code are the most significant part of the development process. Even though the code is working for its intended purpose, comments can tell the intent and the reasons for a developer or collaborator to understand not only how a piece of code works but also why certain decisions were taken. Comments are very important for rather complex, evident parts of the code, thus clearly for others, maybe even the writer themselves, what the logic behind the code is, without having to disassemble the entire code base. The utility of code comments extends beyond a single developer to improve communication among the members and, importantly, to ensure long-term maintainability of software.

1.1.1. Enhancing code readability

Good, clear comments make code much easier to read. For the experienced developer, sometimes it is even obvious enough to understand code briefly, but for the rest of the team or a newcomer to the project, this was not the case. Finally, the comments break down difficult functions or obtuse logic into understandable explanations. This brings clarity to the developer reading the code quickly so that they don't need to trace every variable and function call, thus enabling the comments to ensure that the code remains accessible and understandable to all stakeholders.

1.1.2. Facilitating code maintenance

Applications do evolve with time. Code is bound to change- from minor corrections in bugs to more architectural overhauls. Good comments make the code much easier to keep up with because comments do not merely describe what it does but why it's done that way. This might be particularly crucial if the original logic is complicated or if a trade-off was given to optimize performance. Comments provide essential insights to future developers, even to the original coder, which helps reduce the amount of time spent trying to decipher why the code was created, making updates and modifications smoother and less prone to error.

1.1.3. Promoting team collaboration

In the collaborative development environment, comments are but one of the keyways that team members relate to one another and the code being developed. When multiple developers must collaborate on code bases, each with potentially disparate familiarity with the code, everyone needs the ability to understand each other's thinking when developing each piece of code. Comments allow team members to put on paper their thought processes, which may make it easier for others to read, refactor, or build on top of their code. The code becomes even easier to understand when learning newcomers join the project because comments immediately show how that piece of software works and why decisions were made.

1.2. Organization of the paper

This paper discusses the importance of program comprehension and challenges in producing quality comments for code. It further reviews related work on code comment generation from template-based methods to modern encoder-decoder frameworks. A newly proposed method, codex Comment, is then discussed, including architecture details and discussion about the approach on hybrid code representation. This paper discusses the experimental setup consisting of datasets and evaluation metrics together with their results, comparing it with baseline methods. It concludes with paths to further research.

2. Architecture diagram

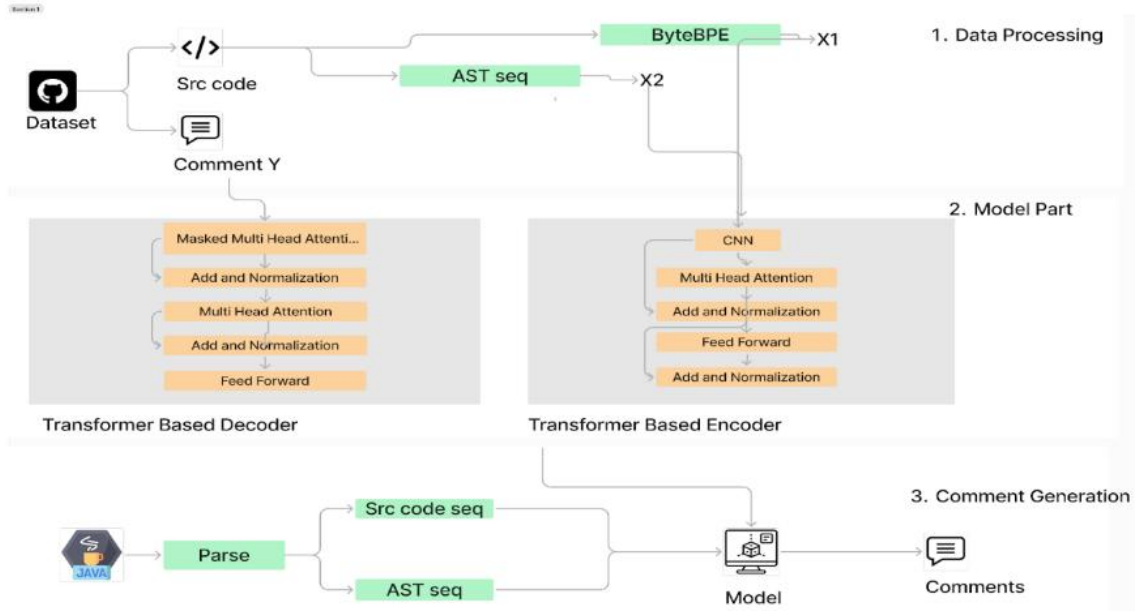


Fig. 1: Architecture Diagram.

2.1. Data processing part

The Data Process Part is important to change the raw source code into a form that can efficiently be processed by the model. It begins with the formation of a Hybrid Code Representation by integrating two critical layers of information: Lexical-Level Information and Syntactic-Level Information. Lexical-Level Information refers to extracting the tokens from the code directly, which includes the fundamental syntax and keywords used in the programming language. These tokens allow the model to capture the basic structure as the sequence of operations within the code. For the Syntactic-Level Information, this will come out of the Abstract Syntax Tree, which is a hierarchical representation of the code structure. The Abstract Syntax Tree, besides showing relationships between different code elements- for example, functions, variables, and control structures- would indeed give the model more insight into code's logic and flow. Following this, the stages of pre-processing data will be implemented, which comprise tokenization, normalization, and generation stages to clean and format the code to make it consistent with the preparation required before entering the subsequent analysis stages.

2.2. Model part

This core processing element in Model Part uses the architecture of the Transformer, which is capable of handling sequential data. The core architecture of this model can be established to contain an Encoder-Decoder structure through which the Encoder essentially uses the hybrid code representation as its input and generates context vectors that include the information related to the code. These context vectors form the basis for generating insightful comments. To maintain the sequence and flow of tokens in the code, Codex Comment relies on positional embeddings, thereby enabling the model to have insight into how the elements might appear in a sequence and not disrupt the integrity of the logic in the code. In addition, the approach through Fusion Methods at the Encoder stage applied with lexical and syntactic information ensures a total representation of both surface syntax and the deeper structure of the code. This fusion lends immense significance to the comments that the model produces, rendering them valid descriptions of the function and behaviour of the code. Other refinements derived from the Transformer architecture are obtained from the attention mechanism: it allows the model the ability to focus on certain segments of the code and better capture the nuances in the complex relationships that exist within the code.

2.3. Comment generation part

Comment Generation Part The process of transforming processed data into comments in human readable codes. It begins with a procedure called the Decoding Process, whereby, in this scenario, the Decoder uses context vectors produced by the Encoder in generating a sequence of tokens that comprise the final comments. These comments, owing to their reflection of the functionality and intent of the code they describe, are indispensable for developers. For quality control, Codex Comment includes a Quality Control mechanism that evaluates and fine-tunes the output so that the quality comes up improved. This may be done using predefined criteria or additional training data in fine-tuning the generated text. Performance metrics like BLEU, ROUGE, and METEOR score s determine comment quality and relevance for further intensive evaluation. Being automated, human evaluation is typically performed at various stages to validate the effectiveness of produced comments and their clarity. The multi-layer approach in this way produces high-quality comments and effectively improves the readability of code, raising the developer's productivity.

3. Data flow

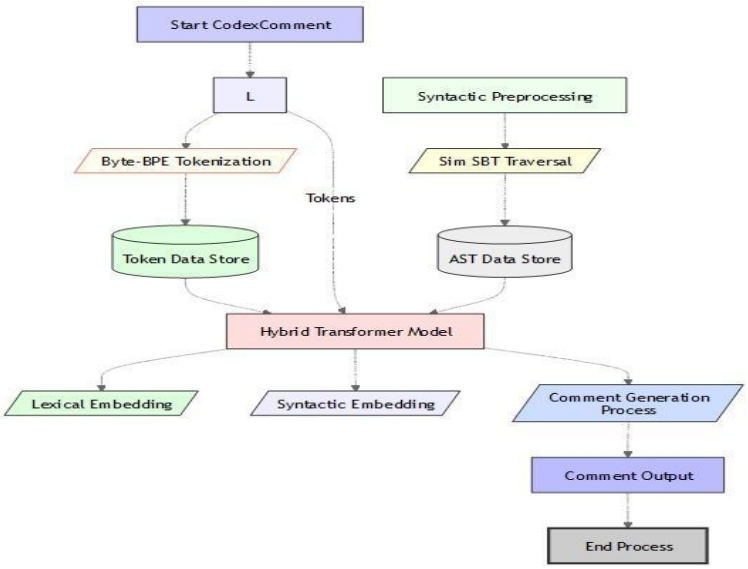


Fig. 2: Data Flow Diagram.

3.1. Input processing and tokenization

The Codex Comment process starts by accepting input code or text, which is represented in the workflow as "L". First processed through Byte-Pair Encoding (BPE) Tokenization. BPE is a technique used widely and spreads out the input in units of subwords or tokens. It easily manages large vocabularies and out-of-vocabulary words, which is important in dealing with complex codebases and varied terminology. These tokens are then placed in a Token Data Store so that they can be retrieved later for processing. While the input is going through Syntactic Preprocessing, syntax-specific information about the input is captured: keywords, operators, functions, and structural components of the program. This preprocessing is vital as it equips the input with syntactic knowledge, such that Codex Comment captures the structural pattern associated with the code.

3.2. Syntax tree analysis (sim SBT traversal)

After tokenization and syntactic preprocessing, Codex Comment applies Sim SBT to the syntactically pre-processed data. Sim SBT organizes data into a structured format like an Abstract Syntax Tree that helps Codex Comment understand the hierarchical relationships within the code. It will distinguish functions, loops, and conditional statements that represent how each component is related to others. This hierarchical structuring is important to produce meaningful comments because it reflects the logical structure of the code. The traversed data would then be stored in a Data Store named AST, which retains the syntactic information extracted and enables the Codex Comment to access the structure-oriented data necessary in subsequent stages.

3.3. Hybrid transformer model

After the lexical information has been processed from the tokens and the syntactic information has been accessed from the Abstract Syntax Tree, the two will be inputted into Codex Comment's Hybrid Transformer Model. The model is "hybrid" because it combines two streams of information: word-based lexical information and structure-based syntactic information. In this way, the Hybrid Transform Model can interpret the input in a more holistic sense than only being able to understand what words are present but how those words go together and how the words interact. Output for two types of embeddings: Lexical embedding captures nuances and meaning on a word level, and syntactic embedding of the structure and logical flow of the input. The final accurate comment generation would be represented with contextual and sensitive-to-code structure embeddings of the text.

3.4. Comment generation and output

Finally, in the final step, Codex Comment takes in lexical and syntactic embeddings to be engaged in the Comment Generation Process. This then generates a comment that is coherent with both word-level and structure-level understandings. In other words, feeding in both kinds of understandings to Codex Comment can aid it in generating comments that are both contextually pertinent and reflect a sense of logical structure and even a purpose behind the code. Then this comment is generated into Comment Output, which is a user-ready insightful annotation of the original code or input.

4. Result and discussion

METHOD	BLEU_1(%)	BLEU_2(%)	BLEU_3(%)	BLEU_4(%)	METEOR(%)	ROUGE_L(%)
DeepCom	49.023	44.140	38.265	35.216	25.183	52.175
Hybrid-DeepCom	54.056	45.046	40.336	37.397	27.383	54.331
Transformer	55.624	46.295	41.574	38.692	29.056	55.263
CodePtr	59.506	51.107	46.386	43.371	31.382	62.761
Seq2Seq	45.016	40.625	36.162	34.024	23.695	50.462
Seq2Seq with atten	46.526	41.526	37.812	35.041	24.534	51.842
GPT-2	47.915	41.253	37.593	35.301	26.887	53.398

The performance metrics for various methods of code comment generation show that there is a clear hierarchy among their effectiveness concerning the former winning all metrics measured: CodePtr ranked at the top with impressive BLEU scores ranging from 59.506% for BLEU-1 to 43.371% for BLEU-4, which hints at its ability to generate good comments. CodePtr achieved a METEOR score of 31.382%, which indirectly translates to the degree of similarity between the produced code and the decision made by human judgment in terms of comment quality and a strong ROUGE-L score of 62.761% that is indicative of its effectiveness in capturing the longest common subsequence between generated and reference comments. Continuing in the lines of CodePtr, the Transformer model also succeeded with outstanding results, achieving a minimum BLEU of 55.624% (BLEU-1) and a maximum of 38.692% (BLEU-4), along with a METEOR score of 29.056% and a ROUGE-L score of 55.263%, so it indicates its reliability in producing meaningful comments. The Hybrid-DeepCom also performed relatively well by BLEU scores of 54.056% (BLEU-1) to 37.397% (BLEU-4), a METEOR score at 27.383%, and ROUGE-L at 54.331%. It appears that in comparison to CodePtr and the Transformer, this seems plausible but not on par with the leaders. In contrast, DeepCom performed poorly and had BLEU scores ranging from 49.023% to 35.216%, a

METEOR score of 25.183%, and a ROUGE-L score of 52.175%. This indicates vast scope for improvement. Among the variants, the model with attention was the least effective and had BLEU scores of 45.016% and METEOR scores below 25%, which indicates that it performs poorly in generating high-quality comments. Overall, the results point out that CodePtr is indeed the best of its kind in the world of code comment generation and also points out the varying effectiveness of the other methods evaluated in the experiment.

According to the sources, however, this area still has a gap in lots of research about the automatic generation of code comments, even if great achievements have been given by models like CodexComment. As with CodexComment applying a hybrid representation of code that incorporates both lexical and syntactic information, it has not overcome the limitation of processing arbitrarily complex code structures and semantics. This calls for more sophisticated representation techniques that are more sensitive to deeper semantic relationships than what is visible on the surface. Future work might be the integration of dynamic execution information and program control flows for better understanding the code behaviour and intent. It seems that a new line of research is still lacking, dataset bias and weakness.
The problems refer to difficulties arising from the imbalanced datasets on which the models are trained, extremely concerning rare words or code constructs. This later results in poor generalization to unseen code and invalid comments from the models. Further research should be on developing more representative datasets. Some other techniques, including data augmentation and synthetic data generation, would be used to enhance the diversity or size of training data that may help in developing robust and reliable models.

All resources, thus, certainly point towards a greater necessity of human-centred and more comprehensive evaluation metrics in code comment generation than the currently widely applied BLEU or METEOR metrics. The latter metrics seem to be more focused on lexical similarity rather than trying to provide any piercing insight into the quality of comments from a developer's point of view.
Future work should focus on evaluation methods that are nearer to human judgment in the form of comment naturalness, relevance, and understandability. This will demand the integration of human evaluation or designing new metrics that target code comprehension and maintainability particularly. Such problems will fill gaps in research and take the field forward in generating high-quality, human-like comments in code, which will further intensify program comprehension and developer productivity.

5. Conclusion

The sources described how the CodexComment is a very promising innovation in the development of automatic code comment generation. High-quality comments efficiently improving program comprehension can be generated through the combination of hybrid code representation with the Transformer model and techniques such as Byte-BPE and Sim SBT. Fundamentally, research conducted here puts a big stress on realizing existing shortcomings to improve accuracy, generalizability, and usability of tools of this kind.

6. Future scope

Future work on code comment generation will focus on several key areas to increase the effectiveness and utility of the tool. More advanced code representations will be considered where research will be directed to explore deeper semantic relationships in code. Beyond simple lexical and syntactic information, this may include dynamic execution data, control flow analysis, or further contextual information that will give a richer understanding of code behaviour and intent. It tries to generate more accurate code representations, which then, through such a representation, can better comment on the underlying logic and purpose.

Another significant area of future work is concerning balanced and representative datasets. Most of the datasets currently in use present biases or representativity issues, so the model may break down in generalizing to other coding situations. Researchers want to generate diverse and comprehensive training datasets by methods for data augmentation, synthetic data generation, and many more. The models resulting from this approach should improve on a substantially broader scope of code and scenarios, ultimately resulting in better performance and more robust comment generation for unseen code.

In the future, human-centered metrics will be crucial for evaluating the generation of code comments. The nature of the present metrics is predominantly aligned with automatic evaluations and may not correspond well to the judgment of humans. The future work will be on newer evaluation criteria based on the reasons for naturalness, relevance, and understandability of comments as perceived by the developer. This includes consideration of integration of human feedback into evaluations or metrics customized to enhance code comprehensibility and maintainability.
Also, the possibility of the scope of expansion of code comment generation tools like CodexComment to other programming languages is very fascinating. Even though the studies concentrate on Java, expanding these approaches to more languages could be considerably useful. The model and the training dataset would have to be adapted for each programming language so that such tools impact and give meaning across different contexts of software development.

Last but certainly not least, integration into real-time coding environments such as IDEs is highly promising. With it, real-time support for high-quality comment generation at the time of coding will improve productivity and enhance code

quality. Future directions will contribute to developing more powerful tools that enhance program comprehension, streamline developer workflows, and ultimately improve the overall efficiency of software development.

References

[1] S. D. Fabiyi and O. Ajibuwa, "Automatic Code Commenting in Integrated Development Environments Based on Indirect Interaction with Chatbots," 2023 International Scientific Conference on Computer Science (COMSCI), Sozopol, Bulgaria, 2023, pp. 1-4, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10315818&isnumber=10315795. https://doi.org/10.1109/COMSCI59259.2023.10315818.

[2] Y. Park, A. Park and C. Kim, "ALSI-Transformer: Transformer-Based Code Comment Generation With Aligned Lexical and Syntactic Information," in IEEE Access, vol. 11, pp. 39037-39047, 2023, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10105196&isnumber=10005208. https://doi.org/10.1109/ACCESS.2023.3268638.

[3] M. A. E. Kilic and M. F. Adak, "Source Code Summarization & Comment Generation with NLP : A New Index Proposal," 2024 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), Istanbul, Turkiye, 2024, pp. 1-6, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10550711&isnumber=10550449. https://doi.org/10.1109/HORA61326.2024.10550711.

[4] S. Saranya, S. Devi, R. M. Suruthi, N. M and S. B, "Comment Generator for Java using Deep Learning," 2023 International Conference on Intelligent Systems for Communication, IoT and Security (ICISCoIS), Coimbatore, India, 2023, pp. 50-54, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10100539&isnumber=10100336. https://doi.org/10.1109/ICISCoIS56541.2023.10100539.

[5] F. Mu, X. Chen, L. Shi, S. Wang and Q. Wang, "Developer-Intent Driven Code Comment Generation," 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), Melbourne, Australia, 2023, pp. 768-780, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10172531&isnumber=10172342. https://doi.org/10.1109/ICSE48619.2023.00073.

[6] B. Wei, "Retrieve and Refine: Exemplar-Based Neural Comment Generation," 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), San Diego, CA, USA, 2019, pp. 1250-1252, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8952536&isnumber=8952167. https://doi.org/10.1109/ASE.2019.00152.