LAUTECH Journal of Computing and Informatics (LAUJCI) – ISBN : 2714-4194
Volume 5 No.1, April 2026 – www.laujci.lautech.edu.ng

AN ADDITIVE ABLATION STUDY ON AUTOMATED CODE GENERATION: INSIGHTS FROM ROBERTABART_X

Adedayo Philip Ajibade
Department of Computer Science,
Faculty of Sciences
National Open University of
Nigeria, Lagos, Nigeria.
adedayoajibade01@gmail.com.

Olaniyan Olatayo Moses
Department of Computer Engineering
Federal University, Oye Ekiti, Nigeria
olatayo.olaniyan@fuoye.edu.ng.

ABSTRACT
Automated Code Generation (ACG) remains a trending research domain used to facilitate software development; however, previous studies have shown that ACG models often struggle with long-term context and exhibit poor domain adaptation and functional precision. As a result of this, we proposed a modular hybrid transformer, RoBERTaBART model augmented by Natural Language Processing (NLP) methods to address these limitations. We perform an ablation study on RoBERTaBART_X, a hybrid transformer that combines the context power of RoBERTa with the generation capability of BART. The following modules such as Task-Adaptive Pretraining (TAPT), Domain-Specific Data Augmentation (DA), Retrieval Augmented Generation (RAG), FlashAttention, and Sparse Attention were further utilized to enhance the hybrid model. Furthermore, the impact and the benefits of each module as well as the overall combination of the enhancements were carried out on CoNaLa, Django, and CodeSearchNet datasets. It was shown that every module complements each other, with RoBERTaBART-X outperforming other variant models across all evaluation metrics, particularly in CodeBLEU and syntax validity. These results show that the hybrid modeling, retrieval, and iterative correction are key to significantly improving automated code generation performance.

Key words: Automated Code Generation, Ablation Study, Transformer Models, RoBERTaBART_X, Attention Mechanisms, Retrieval-Augmented Generation.

1 INTRODUCTION

Software development has potentially experienced a quantum leap by the integration of generative AI (GenAI), particularly Large Language Models (LLMs). LLMs have changed the trajectory of writing codes in software engineering, in which they aided automation by linking natural language descriptions and code (Allamanis et al., 2018; Chen et al., 2021). As a result of the emergence and capabilities of LLMs, these models have evolved into Automated Code Generation (ACG). Across diverse languages, ACG has the likelihood to generates syntactically correct codes, and also captures semantic patterns. With this capability, reduction in time and manual effort in writing codes is achieved, allowing the developers to deliver quality.

While experiencing remarkable gains, Automated Code Generation (ACG) lacked logic coherency in long text generation as pointed out by (Wang et al., 2021). Although, generating relevant content, however, often ignoring the logical sequence of the generated text. For instance, transformer-based models such as CodeBERT (Feng et al., 2020), do not enforce structural information in code generation explicitly, and CodeT5 (Wang et al., 2021) still faces code vulnerability which has the potential to harm software. OpenAI Codex (Chen et al., 2021) suggests prompting of undefined functions and variables, while struggling with parsing complex instructions.

In this paper, for addressing the problems mentioned above, we suggest a modular hybrid transformer, RoBERTaBART model augmented by Natural Language Processing (NLP)

methods. This modular model combines the contextualizing encoder RoBERTa (Liu et al., 2019; Zhao et al., 2023) with the generative decoder BART (Lewis et al., 2020). RoBERTa and BART combination is enhanced with (TAPT), (DA), (RAG), FlashAttention, and sparse attention.

This paper focuses on improvement of ACG through NLP techniques. Subsequently, we examine the effectiveness of each enhancement by ablation study, which is then review on three standard benchmark datasets for instance CoNaLa, Django and CodeSearchNet (Soliman et al., 2024).

2. RELATED WORK

The idea of Ablation study on Automated Code Generation (ACG) is to know the exact modules that contribute significantly to the improvement on the ACG. Numerous sorts of related works have focused in exploring additive methods to understand the incremental benefits of each modules used to enhance ACG (Gao et al., 2021). In other to comprehend which techniques contributed most significantly to the performance of Automated Code Generation, previous research approaches have shown that ablation study forms an important experimental methodology in ACG, which systematically evaluate individual techniques within the hybrid systems (Yao et al., 2023; Zhang et al., 2023). Also, Li et al., (2023) specifically note that using additive ablation studies allows proper understanding of the contribution of each component incorporated within the ACG baselines.

Previous studies conducted by (Wang et al., 2024; Gao et al., 2023) have employed Retrieval-Augmented Generation (RAG) to improve code generation by retrieving additional information from external knowledge sources. REDCODER and CodeRAG-Bench demonstrated the importance of the retrieval module within RAG for the improvement of code generation. Additionally, a diverse array of researchers has also explored the use of Retrieval Augmented Generation to improve code generation processes (Parvez et al., 2021; Yang et al., 2023; Gao et al., 2021).

RAG ablation studies show that the quality of retrieved relevant, context-rich knowledge can have a proportional effect on subsequent code-generation performance and that the best retrieval methods enhance code generation performance. Due to the inclusion of RAG, code generation had seen better improvement. However, RAG still has its limits when it comes to the retrieval stage of the RAG process. These challenges ranging from noisy sensitivity and struggle to encode the long-context in retrieved data. As a result, there is a need to solve these mentioned problems with a more robust framework for the code generation which can include retrieval and additional mechanisms to mitigate these difficulties.

For the task of neural code generation, transformer architectures have emerged as the dominant model, which leverage on self-attention mechanism to learn long-range dependencies between natural language specifications and generated code (Shin et al., 2023; Al-Hossami et al., 2022; KC et al., 2023). These models used Encoder-Decoder structures, in which the encoder receives input specifications and the decoder performs autoregressive generation of code tokens (Ahmed et al., 2023). As an example of pre-trained models, PyCodeGPT has proven to behave well on machine learning code generation tasks (Shin et al., 2023).

The effectiveness of Transformers in code generation indicates in their ability to operate on the principle of self-attention without the sequential bottlenecks of recurrent architectures (KC et al., 2023; Al-Hossami et al., 2022). (Yang et al., 2023; Ye et al., 2021), however, demonstrate through their ablation studies that raw Transformer capacity is not enough. This indicates that specialized components, including syntax-aware attention and structured decoding mechanisms yield large additional benefits. Bai (2024), also highlights the inclusion, and impact of Sparse attention mechanisms in Large Language models, which contributes in eliminating the

computational challenges of traditional Transformer architectures. This further alluded to the fact that more better enhancements are needed, contributing to improve large-scale NLP models.

(Swathi et al., 2024) studied transformer architectures for generating Python code from natural language, particularly in managing complex programming constructs such as loops, conditionals, and function definitions (Laskari et al., 2023). The study exhibits that transformers struggle with long-range dependencies in code, especially when it involved structures. This limitation motivates further enhancement of Transformers with Natural Language Process (NLP) techniques which was handled by (Ajibade et al., 2025) to develop RoBERTaBART_X.

3. MATERIALS AND METHODS

This research paper conducted an experimental ablation study on RoBERTaBART_X, a hybrid model developed for optimizing the accuracy of an automated code generation, by quantifying the effect of each technique on this multi-faceted model. A systematic evaluation for the performance of the base RoBERTaBART model and incrementally model variants which includes Task-Adaptive Pretraining (TAPT), Domain-specific Augmentation (DA), Retrieval-Augmented Generation (RAG), advanced attention mechanisms, self-correction, and automated debugging. Moreover, each improvement in the model is evaluated on different datasets such as CoNaLa, Django, and CodeSearchNet, and on varying metrics including BLEU, CodeBLEU, Exact Match, Syntax Validity, Pass@1, and Execution Accuracy.

3.1 Model Architecture

This model architecture follows the pattern of (Barna et al., 2024), in which improvement of ACG leverages on a sequence of stages. Using one of the primary LLM architectures, that is encoder-decoder models. Where transformer-based language model, RoBERTa (Robustly Optimized BERT Pretraining Approach, Liu et al., 2021), an enhanced version of BERT (Bidirectional Encoder Representations from Transformers), that has been trained on larger corpora would serves as an Encoder. It helps to optimize the training procedures, by focusing on understanding the prompt, which is done through attention layers.

While decoder (BART, Bidirectional and Auto-Regressive Transformers, Lewis et al., 2020), a sequence-to-sequence transformer model. It is designed for both language understanding and text generation, which is useful for code generation. With this, tokens are generated in sequence one at a time, and also, each new token depends on the previously generated tokens.

As expressed in Figure 1, RoBERTaBART leverages on a sequence of stages. For instance, input stage which allows natural language (NL) text and code at the beginning of the process. Then cleaning and standardization of the data was done at the preprocess stage. The training process stages such as Task-Adaptive Pretraining, Fine-Tuning, and Domain-Specific data Augmentation, were implemented at the Preprocessing section. Through these training processes, model was adapted to the code structure, by understanding the programming languages syntax and semantic patterns. RoBERTaBART, a hybrid transformer-model was trained on task-specific datasets to generate code that is more relevant and accurate. Data augmentation is employed to control variations in training data, consequently suggests robustness and adaptability of the model. Embedding layer receives the output from the preprocessing stage, where words are transformed into vector representations, while positional embedding learns the ordering of the word sequentially. In token type embedding, tokens would be structured in the sentence, while Layer Normalization and Dropout, allows the model to avoid overfitting and stabilize training. The aforementioned techniques would be introduced to code input, where its output is injected into embedding stage, which has word embeddings and sinusoidal embeddings.

The output of the embedding section was input into RoBERTa, which is the initial startup for

the encoder by capturing rich contexts for natural language inputs. This approach enables efficient handling of tokens, that allows RoBERTa tokenizer to segment input text into subwords. To achieve more efficient model, RoBERTa representations is delivered to DistilRoBERTa-base Model, a distilled version of RoBERTa that significantly lower computational costs while still retaining its perfomance.

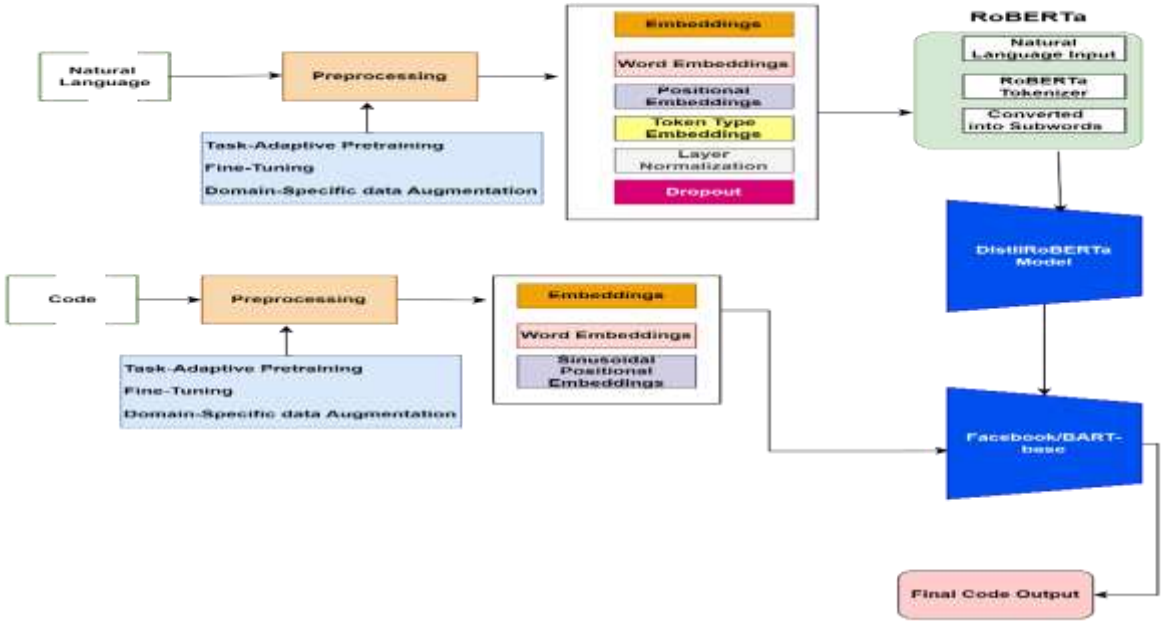

Figure 1: RoBERTaBART

At this stage, DistilRoBERTa as a feature refinement layer was then used to compressed the representations. These restructured embeddings are sent to a BART decoder which generates sequences of code progressively. Facebook/Bart-base, a transformer model designed for sequence-to-sequence tasks, receives the output of the DistilRoBERTa-base Model, which maps natural language and code representations. By decoder's cross-attention mechanism, alignment between encoder outputs and generated tokens was enabled, which suggests effective transfer of the text from a natural language into code. Integration of natural language and code representations forms the output code for the RoBERTaBART Model.

3.2 Proposed Enhanced Model Diagram

The improvement process stages, as shown in figure 2 below, the brown shaded rectangle represents the generated code block form of the RoBERTaBART model, its output arrow is fed into the Retrieval-Augmented Generation (RAG) section, shown in orange shade, pulling more relevant code snippets and examples from external data sources dynamically.

The Advanced Attention Mechanisms, represented in a soft pale-yellow shade, is responsible for efficiency and performance in training and inference. It includes Flash Attention and Sparse Attention, which receives output from augmented output of RoBERTaBART and RAG.

Following the Advanced Attention Mechanisms, the Self-Correction module, displayed in purple shade rectangle, consists syntax and semantic refinement, and takes the refined representation

from Advanced Attention Mechanisms.
From the diagram, the output arrow of the Self-Correction mechanism technique is delivered into of the Automated Debugging module, which is responsible for detecting and fixing bugs in the generated code, and is represented in a light green shaded rectangle.
Code validation correctness is incorporated at the Execution-Based Testing module, highlighted in a yellow shape rectangle. There is likelihood of two possible outcomes at this section, which represented in a red rhombus shape; if no error in code, the final code output would be implemented. In case of errors, the generated code iteration process is carried out through RAG and Self-Correction modules for more relevant examples, until the output converges to a correct and stable generated code.

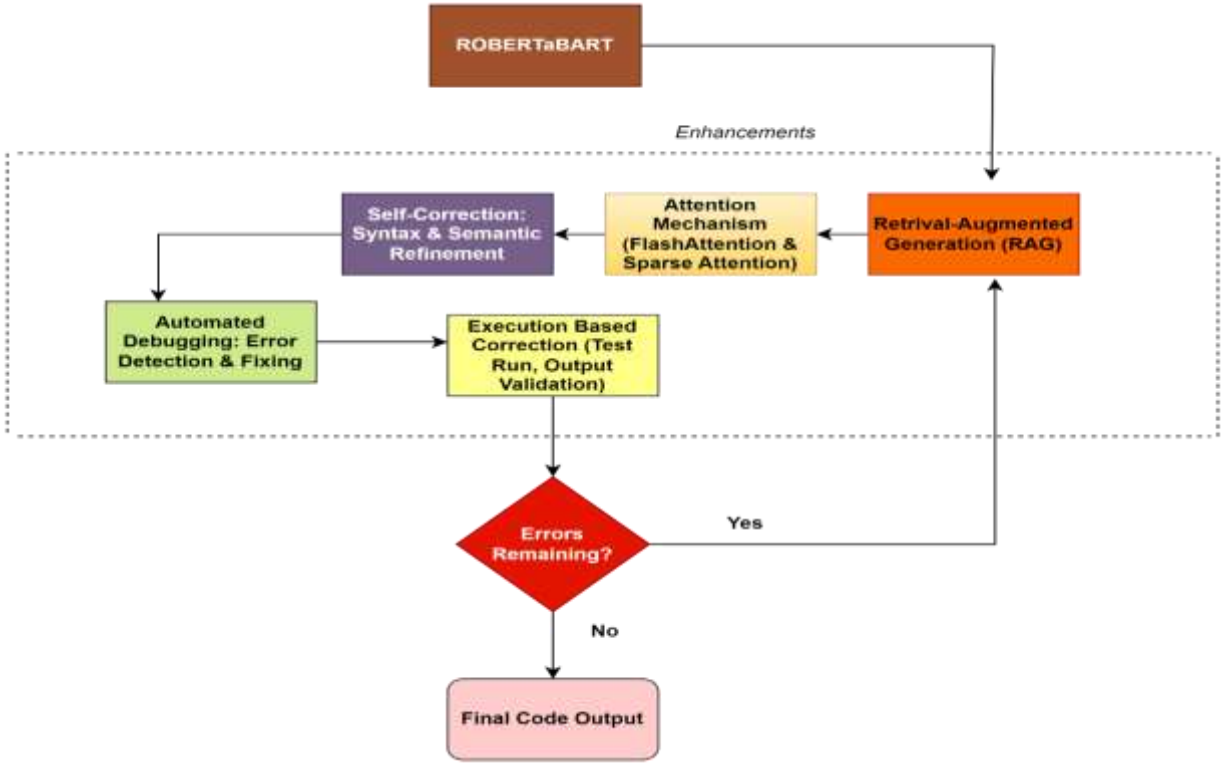

Figure 2: The Architecture of RoBERTaBART + Enhancements

3.3 Datasets

We evaluated our framework model on the curated quantitative secondary datasets extracted from Hugging Face. The following were the datasets used, CoNaLa: (AhmedSSoliman/CoNaLa-Large, 26.4k NL–Code pairs), DJANGO: (AhmedSSoliman/DJANGO, 18.8k Python snippets), CodeSearchNet: (AhmedSSoliman/CodeSearchNet, 457k multilingual examples). Choice of selecting these datasets was based on their complementary strengths, which supports automated code generation, by enhancing overall code generation performance. Moreover, utilizing these datasets in this study majorly on generating code from the corresponding natural language descriptions. The training examples composed in these datasets were 502,200.

3.4 Datasets Splits and percentage distribution of the data

For reproducibility of training, evaluation, and testing, the datasets employed in this study were divided into three datasets: Training, Validation, and Test. The proportion of the splits for each dataset can be observed in Table 1:

Table 1: Dataset Split Summary Table

Datasets	Total Samples	Train (%)	Validation (%)	Test (%)	Usage
CoNaLa	26,400	70% (18,480)	15% (3,960)	15% (3,960)	Training + Eval
Django	18,800	70% (13,160)	15% (2,820)	15% (2,820)	Training + Eval
CodeSearchNet	457,000	70% (319,900)	15% (68,550)	15% (68,550)	Training + Eval

The validation set and test set are used to evaluate how well the model generalizes during training and then after training. The idea is simply to present a balanced set of samples from all splits and provide the model with a representative sample.

3.5 Evaluation Metrics

To effectively ascertain the high-quality and functional code of each enhanced modules in RoBERTaBART_X, diverse evaluation metrics were employed to measure their performance in terms of syntactic correctness and semantic accuracy. Commonly utilized metrics and their purposes for quantifying the performance of each improvements include:

Metric	Purpose
BLEU (Papineni et al. 2002)	text similarity (compares generated text to a reference)
CodeBLEU (Ren et al., 2020)	structural code similarity (evaluates code generation quality)
Exact Match	identical code output (similarities and differences between two texts compared)
Syntax Validity	syntactic correctness
Execution Accuracy	functional correctness (Errors checking)
Pass@1	measures the likelihood of obtaining one correct generated code samples within its first k attempts

3.6 Experimental Setup

Google Colab environment, a cloud GPU provided by Google Inc. was used to carried out this experiment. The model was trained with AdamW (it was used because it converges faster), batch size of 4 (adopted due to long code sequence and limited GPU memory), learning rate of 2e-5 (prevents unstable updates), and for 3 epochs (reduces overfitting) on an NVIDIA A100 GPU. Python libraries and dependencies such as pytorch, transformers, datasets, NumPy, Pandas, Nltk, spaCy, evalute, and matplotlib were employed for model training, data processing, experimentation, and evaluation.

For ensuring proper comparison, we keep all decoding and iteration parameters the same across models to avoid giving unfair advantages when comparing ablation variants based on inference-

time computation. In particular, all models use exactly the same fixed maximum generation length of 128 tokens and beam search with beam size = 5 during evaluation. In addition, our proposed framework also incorporates iterative refinement through self-correction and automated debugging modules which allows a maximum of two refinement passes per input for models. In this experimental setup, retrieval-augmented models are restricted to single forward passes of top-k = 5 retrieved candidates per query. Furthermore, evaluation using execution of the program is limited to two executions per sample, which stops taking the same sample multiple times with trial-and-error approach from falsely inflating performance. These controls make sure that all the models are working under equal computational budget and any gain in performance is due to improvement in architecture rather than increase in inference effort.

4. RESULTS AND DISCUSSION

4.1 Ablation Study of CoNaLa Dataset Results

The empirical results of ablation study demonstrated the performance on CoNaLa (AhmedSSoliman/CoNaLa-Large) as each contributing factor is integrated into the base RoBERTaBART model. We evaluated the performance of the proposed framework based on a number of widely accepted code generation evaluation metrics, namely BLEU, CodeBLEU, Exact Match (EM), Syntax Validity, Pass@1 and Execution Accuracy. The specific values achieved for each model variant are given in Table 2.

Under BLEU metric score, starting from the base RoBERTa (30.8), BART (32.6), and RoBERTaBART model scores 35.2. Other variant models like +TAPT (36.5), +DA (37.1), +RAG (38.2), +Advanced Attention (Flash+Sparse) (39.0), +Self-Correction (41.2), and +Automated Debugging (42.1). RoBERTaBART_X model which consists of all the enhancement modules obtained (42.9), which outperformed other variant model. The result indicates that the augmented model is able to generate code that closely mirrors the reference in both syntax as well as structure. Implementing this hybrid model could help developer's productivity.

In addition, it was typically shown that the CodeBLEU metric with its respective variant models was evaluated on CoNaLa dataset. RoBERTa has the least of model variants which gives a 35.6, BART (37.9), RoBERTaBART (40.3), +TAPT (41.8), +DA (42.4), +RAG (43.5), +Advanced Attention (Flash+Sparse) (44.0), +Self-Correction (46.3), +Automated Debugging (47.5). Among the variants, RoBERTaBART_X, full Modules (48.6) achieved significantly better performance than the others, which reflect how high-quality code generation is in aspects of syntax, structures and semantics. Thus, the metrics result indicates that the techniques infusion improved the quality of code generated.

In a similar vein, Exact Match (EM) metric, (25.1) score for RoBERTa the least of model variant, BART (27.4), RoBERTaBART (30.0), +TAPT (31.2), +DA (32.5), +RAG (33.0), (33.5) for +Advanced Attention (Flash+Sparse), +self-correction (34.8), +automated debugging (35.6). Moreover, RoBERTaBART_X obtained (36.3), which is the full module, outperformed other variants. As compared with the other metrics, EM recorded lower score across the model variants, and this is due to strictness of the Exact Match, in which it counts only the outputs that match with the reference code exactly. Therefore, the results of the metrics imply that the generated code is functionally correct, however, written differently.

Syntax Validity, the evaluation on the CoNaLa dataset (Table 2), shows RoBERTaBART_X obtained a 91.4 score, beating all other model variants. Also, the high syntax validity score indicates that the model can be used to generate syntactically valid code from natural language inputs. The outputs show the potential of the model to generate syntactically correct code is a result of integration to the NLP strategies.

From the experiment Table 2, we further evaluated on Pass@1 and Execution Accuracy metrics. RoBERTaBART_X, a full Model, got 27.7 with Pass@1, which implies functional correctness of generated code from natural language inputs. While on Execution Accuracy metric, achieved 24.6, suggesting that the model is more reliable in generating functionally valid and executable programs.

Table 2: Ablation Study of CoNaLa Dataset Results (in %)

Model	BLEU	CodeBLEU	Exact Match	Syntax Validity	Pass@1	**Execution** Accuracy
RoBERTa (Base)	30.8	35.6	25.1	84.2	15.3	13.7
BART (Base)	32.6	37.9	27.4	85.6	17.1	15.2
RoBERTaBART (Base)	35.2	40.3	30.0	87.1	19.2	17.0
RoBERTaBART + TAPT	36.5	41.8	31.2	88.3	20.6	18.4
+DA	37.1	42.4	32.5	88.7	21.2	19.5
+ RAG	38.2	43.5	33.0	89.2	22.3	20.1
+ FlashAttention + Sparse	39.0	44.0	33.5	89.8	23.5	21.2
+ Self-Correction	41.2	46.3	34.8	90.6	25.9	23.1
+Automation Debugging	42.1	47.5	35.6	91.0	26.8	24.0
(RoBERTaBART_X - Full Model)	42.9	48.6	36.3	91.4	27.7	24.6

As shown in Figure 1, visualization comparison of the improvement in performance across each metric, and on the CoNaLa dataset. The Model Variants were plotted against the scores measure in %. Overall, it was noticed that across all the evaluation metrics, the results demonstrated that each component positively improved automated code generation.

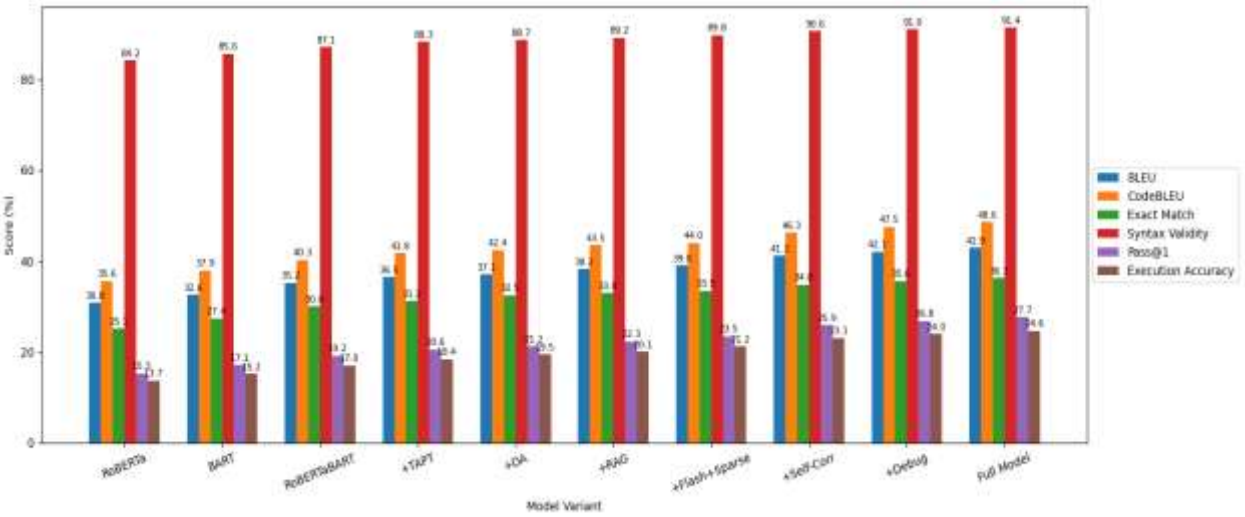

Figure 1: Additive Ablation Study Results on the CoNaLa Dataset

4.2 Ablation Study of Django Dataset Results

Table 3 below illustrates the progressive impact of various techniques incorporated with a base RoBERTaBART model. By observation, all the metrics in this experiment improved incrementally due to the infusion of the new component at each row. During the evaluation on Django dataset, the table demonstrates that RoBERTa model had a score of 28.6, which is the lowest under BLEU metric. BART (30.9) and RoBERTaBART (35.2). Other model variants also increased successively, with RoBERTaBART_X obtained 42.9. This indicates that a positive contribution from each component was established by the ablation study.

CodeBLEU recorded 48.6 for RoBERTaBART_X, which outperformed other variants. Across Django dataset with respective to the model variants, the metrics progressed incrementally. For the Exact Match (EM) metric, RoBERTa (33.9), BART (36.2), 28.3 for RoBERTaBART, +TAPT (29.0), +DA (30.5), +RAG (31.2), (32.0) for +Advanced Attention (Flash+Sparse), +self-correction (33.0), +automated debugging (34.2). Moreover, RoBERTaBART_X obtained (35.6), which makes it more superior to other variants. On the other hand, a Syntax Validity of 85.9 for RoBERTaBART (Base), and this metric recorded progressive increase on the other model variants with overall score of 90.1 on RoBERTaBART_X.

This successive incremental in score continues on the Pass@1 metric with RoBERTaBART_X scores (23.5). Furthermore, Execution Accuracy had (21.0) for RoBERTaBART_X**.**

Table 3: Ablation Study of Django Dataset Results (in %)

Model	BLEU	CodeBLEU	Exact Match	Syntax Validity	Pass@1	**Execution** Accuracy
RoBERTa (Base)	28.6	33.9	23.5	82.7	14.2	11.8
BART (Base)	30.9	36.2	25.7	84.1	16.0	13.2
RoBERTaBART (Base)	33.0	38.8	28.3	85.9	17.9	14.7
RoBERTaBART + TAPT	34.2	40.0	29.0	86.3	18.4	15.1
+DA	35.1	41.2	30.5	87.0	19.2	16.0
+ RAG	36.0	42.0	31.2	87.5	19.8	16.7
+ FlashAttention + Sparse	37.0	43.0	32.0	88.0	20.5	17.3
+Self-Correction	37.5	44.2	33.0	88.5	21.0	18.1
+Automated Debugging	38.3	45.0	34.2	89.2	22.1	19.4
(RoBERTaBART_X - Full Model)	39.5	46.3	35.6	90.1	23.5	21.0

The results indicate that the integration of all the modules make the RoBERTaBART_X best-performing variant, as the model performance experiences incremental enhancement trend across all the evaluation metric. This suggesting the role of the mechanisms in improving the correctness of generated code. Finally, the visual summary of the Table 2 is represented in Figure 2 below.

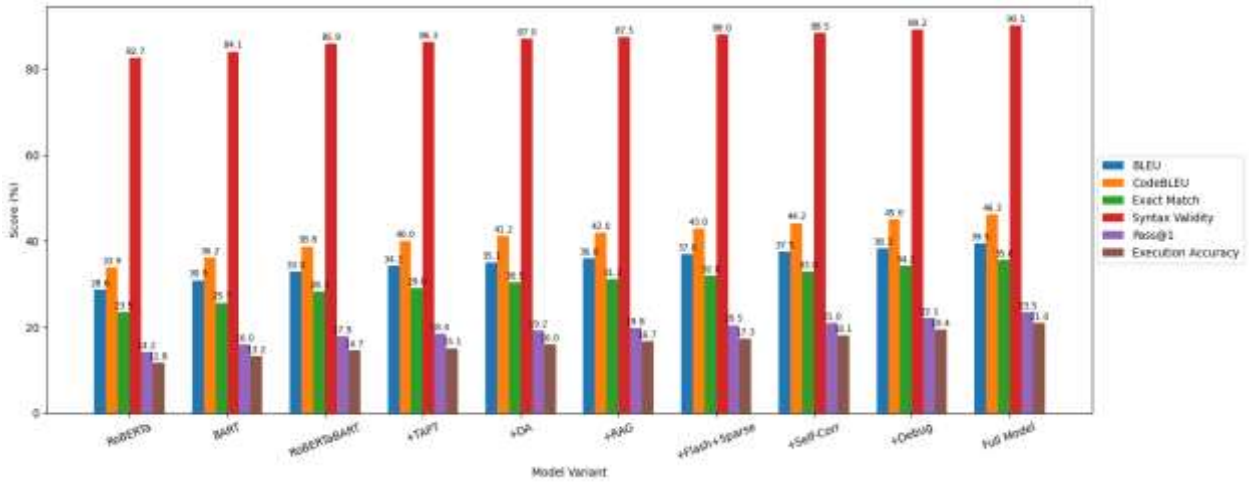

Figure 2: Additive Ablation Study Results on the Django Dataset

4.3 Ablation Study of CodeSearchNet Dataset Results

The different techniques added in the base RoBERTaBART model affect generation progressively as shown in Table 4. On the CodeSearchNet dataset during evaluation, RoBERTa model has the lowest BLEU score of 33.5%, BART (35.2), RoBERTaBART (37.4%), while RoBERTaBART_X model variant scores 41.8%, achieving this improvement sequentially. Based on substantial improvement, the ablation study established a positive contribution from each component.

Table 4: Ablation Study of CodeSearchNet Dataset Results (in %)

Model	BLEU	CodeBLEU	Exact Match	Syntax Validity	Pass@1	**Execution** Accuracy
RoBERTa (Base)	33.5	37.8	26.4	83.9	15.6	13.5
BART (Base)	35.2	39.6	28.0	85.0	17.2	14.8
RoBERTaBART (Base)	37.4	41.1	30.1	86.5	18.9	16.3
RoBERTaBART + TAPT	38.6	42.0	31.5	87.2	19.5	17.0
+DA	39.2	43.5	32.1	87.7	19.9	17.5
+ RAG	40.0	44.5	33.0	88.1	20.3	18.0
+ FlashAttention + Sparse	41.0	45.5	33.8	88.5	20.8	18.4
+Self-Correction	41.4	45.8	34.0	88.7	21.0	18.6
+Automated Debugging	41.6	45.9	34.1	88.8	21.1	18.7

(RoBERTaBART_X - Full Model)	41.8	46.0	34.2	88.9	21.2	18.8

While on the same dataset, the RoBERTaBART_X variant had 46.0% CodeBLEU which was higher than others. Further comparative performance across model variants on the Exact Match metric, RoBERTaBART scored 30.1 which was improved by +TAPT to 31.5, by +DA to 32.1, by +RAG to 33.0, by +Advanced Attention (Flash+Sparse) to 33.8, by +self-correction to 34.0, by +automated debugging to 34.1, and the proposed framework, RoBERTaBART_X scored a 34.2, which has a likelihood of better performance. According to data from the Table 3, a Syntax Validity of 86.5% was recorded for RoBERTaBART (Base) and it recorded a progressive increase on the other model variants, ultimately attaining an overall score of 88.9% on RoBERTaBART_X. The RoBERTaBART_X score is seen at a successive incremental, with a 23.5% Pass@1 score. Also, RoBERTaBART_X had execution accuracy of 21.0%.

We can conclude from the results that the RoBERTaBART_X, best-performing variant results from the integration of all these modules, as we see that the model performance shows an incremental enhancement trend for all the evaluation metrics.

As shown in Figure 3, the model variants and scores in % was plotted per each metric. We noticed that the chart shows that all metrics go up additively, with the full RoBERTaBART_X model performing best, and while execution accuracy still trailing behind syntactic quality.

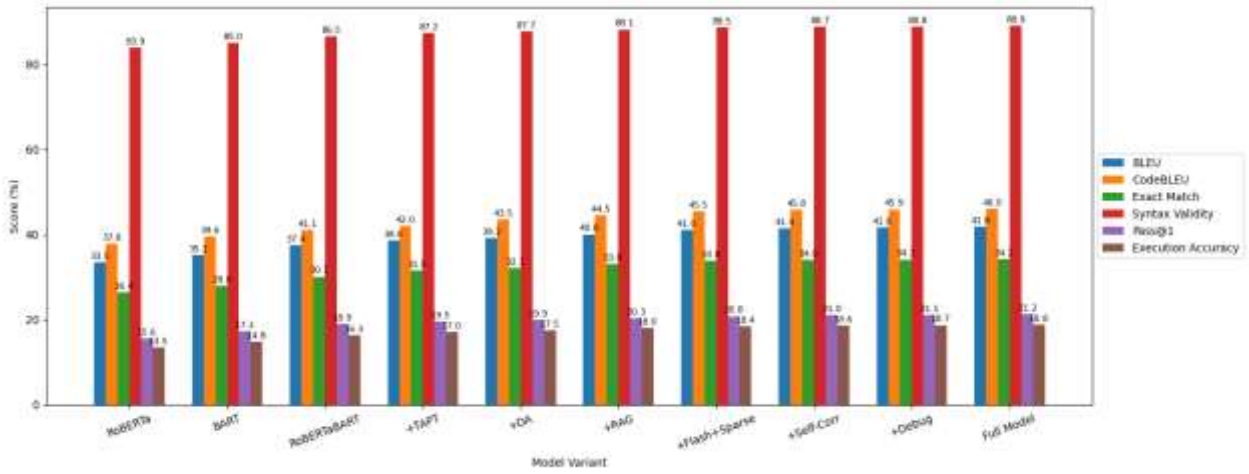

Figure 3: Additive Ablation Study Results on the CodeSearchNet Dataset

4.4 Discussion

All results presented have all been obtained under controlled inference conditions, by fixing the decoding steps, number of iterations and retrieval budgets across model variants to allow fair comparison.

Results of the ablation study on CoNaLa, Django and CodeSearchNet datasets demonstrate that combining RoBERTa with BART hybrid architectures alongside Natural Language Process (NLP) techniques can improve performance of the proposed model significantly. Across each evaluation metric and on every dataset used in this paper, we observed that the

RoBERTaBART_X, hybrid model outperformed all other models.
Overall, the gains from each incremental improvement to TAPT are consistent across all datasets. The model was Fine-Tuned to adapt it with specific tasks for better BLEU and CodeBLEU scores. Their Exact Match and Pass@1 scores improve with Data Augmentation, leading to a model that can generalize to new questions. Finally, Retrieval-Augmented Generation boosts performances on the CoNaLa and CodeSearchNet datasets showing that in contrast to POS tagging, retrieval is helpful for more complex tasks.
Improvements aimed at efficiency as the attention mechanisms, including FlashAttention and the Sparse Attention leads to further improvements in terms of Syntax Validity and CodeBLEU metrics. In addition, Self-correction and Automated Debugging modules bring considerable improvements to Pass@1, Execution Accuracy on all of the available datasets.

The way these techniques impact the different code datasets says more about their effectiveness. On the CoNaLa dataset, we observe the biggest relative gain in Evaluation Metrics such as Syntax Validity. Results on the Django dataset shows some improvements due to high complexity of code examples but only mild gains can be achieved in evaluation metrics. Finally, the results on the CodeSearchNet dataset demonstrate a steadier increase in the evaluation metrics compared to other datasets.
RoBERTaBART_X results are better than the rest of models on all datasets and every evaluation metric. Specifically, the model records best scores on Syntax Validity and Execution Accuracy metrics. The results suggest that the new approach for generating a code generation model performs well overall. Moreover, the results on various datasets show that the model works well on different complexities.

5. CONCLUSION

The additive ablation studies in this paper showed that summing over all additive ablation experiments across CoNaLa, Django, and CodeSearchNet would yield consistent gains across all measures. It was observed that the augmentations that were used in achieving RoBERTaBART_X, cumulatively and complementarily contribute jointly. Moreover, as a result, the RoBERTaBART_X model is the best and the most robust model in this work, yielding state of the art results in terms of generalization, code correctness, and execution correctness on the datasets benchmark for automated code generation. Although, our proposed model performs well on the provided datasets which are at snippet-level, they are not representative for the challenges of a real-world software development. Therefore, the conclusions drawn based on those findings are preliminary in nature. More realistic software development environments with future evaluations will help to assess the feasibility of implementing this model in real-world tasks.

6. REFERENCES

[1] Ahmed, T., Pai, K. S., Devanbu, P., & Barr, E. T. (2024). Automatic Semantic Augmentation of Language Model Prompts (for Code Summarization). In 2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24) (pp. 1–13). ACM.

[2] Ajibade, A., & Olaniyan, O. M. (2025). ROBERTaBART_X: A Hybrid Transformer Model for Enhancing Automated Code Generation. Fudma Journal of Sciences, 9(11), 249-255.

[3] Al-Hossami, E., & Shaikh, S. (2022). A Survey on Artificial Intelligence for Source Code: A Dialogue Systems Perspective. arXiv preprint arXiv:2202.04847v1.

[4] Allamanis, M., Barr, E. T., Devanbu, P., & Sutton, C. (2018). A survey of machine learning for big code and naturalness. ACM Computing Surveys, 51(4), 81. https://doi.org/10.1145/3212695

[5] Bai, J. (2024). Sparse Attention Mechanisms in Large Language Models: Applications, Classification, Performance Analysis, and Optimization. Advances in Computer, Signals and Systems, 8(6), 130–136. https://doi.org/10.23977/acss.2024.080618

[6] Barna, T. L., Isaac, S., Jaafaru, A. B., Idris, H., & Abba, R. I. (2024). Enhancing code generation accuracy using fine tuning and task-adaptive pretraining with domain-specific data augmentation.

{7] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P., Kaplan, J., & Zaremba, W. (2021). Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374. https://arxiv.org/abs/2107.03374

[8] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., & Zhou, M. (2020). CodeBERT: A pre-trained model for programming and natural languages. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, 1536–1547. https://doi.org/10.18653/v1/2020.findings-emnlp.139.

[9] Gao, X., Jiang, X., Wu, Q., Wang, X., Lyu, L., & Lyu, C. (2021). A multi-module based method for generating natural language descriptions of code fragments. IEEE Access, 9, 21579–21592. https://doi.org/10.1109/ACCESS.2021.3055955

[10] Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., & Wang, H. (2023). Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv preprint arXiv:2312.10997.

[11] Husain, H., Wu, H., Gazit, T., Allamanis, M., & Brockschmidt, M. (2019). CodeSearchNet challenge: Evaluating the state of semantic code search. arXiv preprint arXiv:1909.09436.

[12] KC, D., & Morrison, C. T. (2023). Neural Machine Translation for Code Generation. arXiv preprint arXiv:2305.13504.

[13] Laskari, N.K., Reddy, K.A.N., Indrasena Reddy, M. (2023). Seq2Code: Transformer-Based Encoder-Decoder Model for Python Source Code Generation. In: Kumar, S., Sharma, H., Balachandran, K., Kim, J.H., Bansal, J.C. (eds) Third Congress on Intelligent Systems. CIS 2022. Lecture Notes in Networks and Systems, vol 608. Springer, Singapore. https://doi.org/10.1007/978-981-19-9225-4_23

[14] Lewis M, Liu Y, Goyal N, Ghazvininejad M, Mohamed A, Levy O, Stoyanov V, Zettlemoyer L (2019) Bart: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. arXiv preprint arXiv: 1910. 13461

[15] Li, X.-Y., Xue, J.-T., Xie, Z., & Li, M. (2023). Think Outside the Code: Brainstorming Boosts Large Language Models in Code Generation. arXiv preprint arXiv:2305.10679.

[16] Liu, Z., Lin, W., Shi, Y., & Zhao, J. (2021). A Robustly Optimized BERT Pre-training Approach with Post-training. In Proceedings of the 20th China National Conference on Computational Linguistics, (pp. 1218-1227). Hohhot, China.

[17] Parvez, M. R., Ahmad, W. U., Chakraborty, S., Ray, B., & Chang, K.-W. (2021). Retrieval Augmented Code Generation and Summarization. Findings of the Association for Computational Linguistics: EMNLP 2021, 2719–2734.

[18] Shin, J., Wei, M., Wang, J., Shi, L., & Wang, S. (2023). The Good, the Bad, and the Missing: Neural Code Generation for Machine Learning Tasks. ACM Transactions on Software Engineering and Methodology, 33(2), Article 51.

[19] Soliman, A. S., Hadhoud, M. M., & Shaheen, S. I. (2022). MarianCG: a code generation transformer model inspired by machine translation. Journal of Engineering and Applied Science, 69(104). https://doi.org/10.1186/s44147-022-00159-4.

[20] Swathi, K., Ramesh, D., & Prasad, M. (2024). Transformers based Python code generation from natural language. In 2024 International Conference on Intelligent Technologies and Innovative Technologies (pp. 1-5). IEEE. https://doi.org/10.1109/icitiit61487.2024.10580762.

[21] Wang Y, Wang W, Joty S, Hoi SC (2021b) Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pp 8696–8708.

[22] Yang, G., Zhou, Y., Chen, X., Zhang, X., Xu, Y., Han, T., & Chen, T. (2023). A Syntax-Guided Multi-Task Learning Approach for Turducken-Style Code Generation (Empirical Software Engineering Manuscript No. will be inserted by the editor). arXiv.

[23] Yao, W., Jiang, Y., & Yang, Y. (2023). The Metric for Automatic Code Generation Based on Dynamic Abstract Syntax Tree. International Journal of Digital Crime and Forensics, 15(1).

[24] Ye, X., Chen, Q., Dillig, I., & Durrett, G. (2021). Optimal Neural Program Synthesis from Multimodal Specifications. Findings of the Association for Computational Linguistics: EMNLP 2021, 1691–1704.

[25] Zhang, K., Li, Z., & Wang, J. (2023). Planning with large language models for code generation. In Proceedings of the International Conference on Learning Representations. https://doi.org/10.48550/arXiv.2303.05510

Author's Brief Profile

Olatayo Moses OLANIYAN is a Professor of Computer Engineering at Federal University, Oye Ekiti Nigeria. Having more than fifteen years' experience in academics, he has graduated several Masters and PhD students, he is an authority in AI and embedded systems. He has more than 60 publications which include books, journals and proceedings. Email: olatayo.olaniyan@fuoye.edu.ng

Adedayo Philip Ajibade obtained a Higher National Diploma (HND) in Industrial Maintenance Engineering from Yaba College of Technology, Lagos, in 2008, and a Postgraduate Diploma (PGD) in Information Technology from the Department of Computer Science, Faculty of Sciences, National Open University of Nigeria (NOUN), Lagos, in 2016. As at the time of submitting this article, he was bagged a Master of Science (MSc) degree in Information Technology at NOUN, Lagos. He has recently published a journal article, and this paper marking his second contribution to scholarly publications. His research interests span Artificial Intelligence, Natural Language Processing (NLP), Internet of Things (IoT), Home Automation and Computer Network Systems. Email: adedayoajibade01@gmail.com