Transformer-Based Deep Coding Engine for Generating High Level Source Code

Team Members

Namrata Das (THA075BEI026)

Neelam Karki (THA075BEI027)

Rakshya Panta (THA075BEI029)

Ruchi Manandhar (THA075BEI035)

Supervised By:

Er. Dinesh Baniya Kshatri

Lecturer

Department of Electronics and Computer Engineering
Institute of Engineering, Thapathali Campus

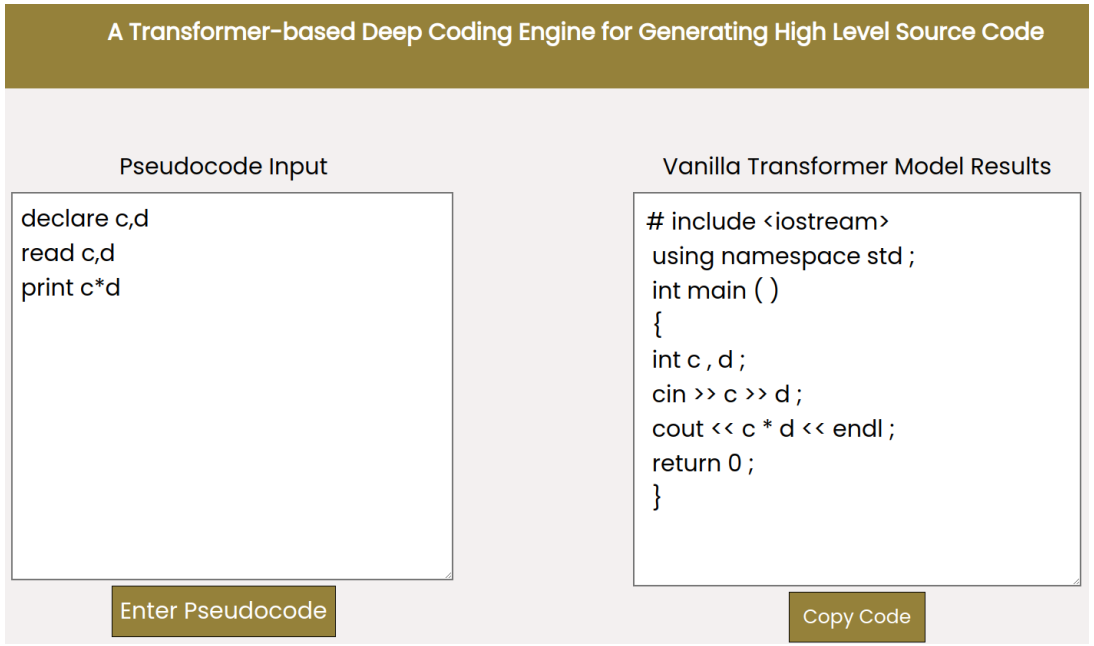
March 2023

Presentation Outline

- Motivation
- Introduction
- Problem Statement and Objectives
- Scope of Project
- Project Applications
- Methodology
- Results
- Discussion and Analysis
- Future Enhancements
- Conclusion
- References

Motivation

Frustrated person stuck in a programming problem



Coding Engine interface performing pseudocode to code conversion

Introduction

- Programming tool to convert pseudocodes to C++ source codes
- Transformer models form the coding engine that converts pseudocode to code
- Algorithmic knowledge and capability of writing pseudocode is expected from the user
- Syntactic knowledge of the programming language is not expected from the user

Problem Statement and Objectives

- Problem Statement
  - Inadequacy of proper tools to synthesize code from pseudocode that perform non-trivial computations
  - Limited benchmarking to judge the functional correctness of automatically generated code

- Objectives
  - To train a vanilla transformer for generating C++ source code via pseudocode of problems related to arithmetic, array, string and sorting operations

  - To perform transfer learning on a pretrained transformer model for better performance on pseudocode to C++ source code conversion

Scope of Project

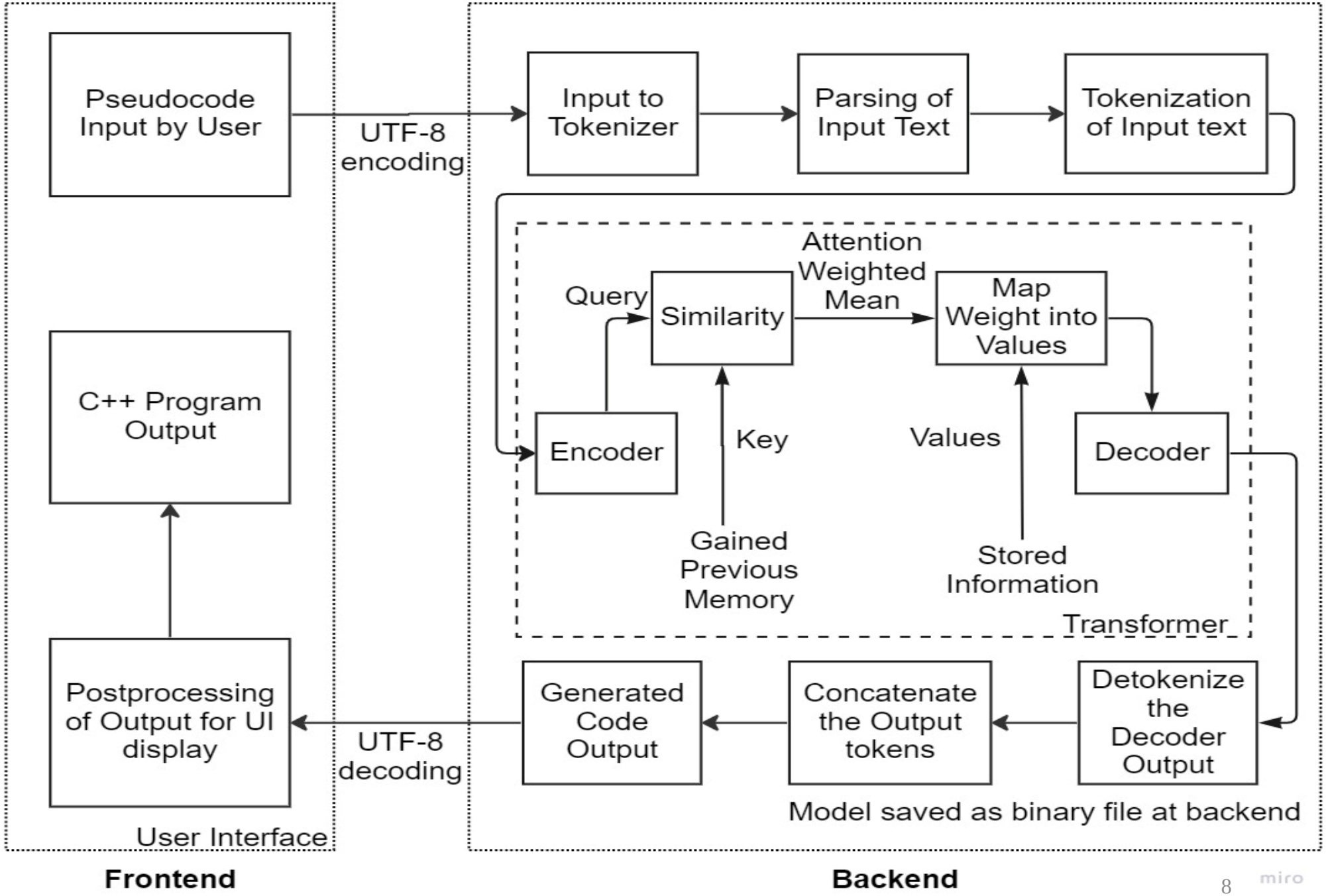
- Project Capabilities:
  - Generates C++ source code from pseudocode
  - Displays generated code in UI
  - Allows copying of the generated source code from UI

- Project Limitations:
  - Does not import all libraries through pseudocode detection
  - Cannot generate source code in languages other than C++
  - Unable to perform software development tasks

Project Applications

- Novice Programmers
  - Coach for beginner coders
- Software Development
  - Assistant to software developers
- Time Critical Demands
  - Coder when programming under time pressure
- Coding contests
  - Automated judge in different coding contests
- Research Institutions
  - Platform for testing automatically generated code

Methodology - [1] (System Block Diagram)



Methodology - [2]
(Working Principle)

- Pseudocode input in text through webpage
- Text input parsed, tokenized and preprocessed
- Preprocessed tokens in numerical form passed to transformer
- Transformer output source code in vector form
- Postprocessing of vectors to obtain source code tokens
- Concatenation of tokens to obtain required source code

Methodology - [3]
(Pre-processing: Tokenization)

- Tokenization of Input Text
  - Splitting input text into constituent words using BERT tokenizer
  - [START], [END], [PAD] and [UNK] tokens are used

- Input Embedding of Tokens
  - Process of mapping tokens to vectors of real numbers
  - Words with same meaning have a similar d-dimensional vector values

Methodology - [4]
(Pre-processing: Positional Encoding)

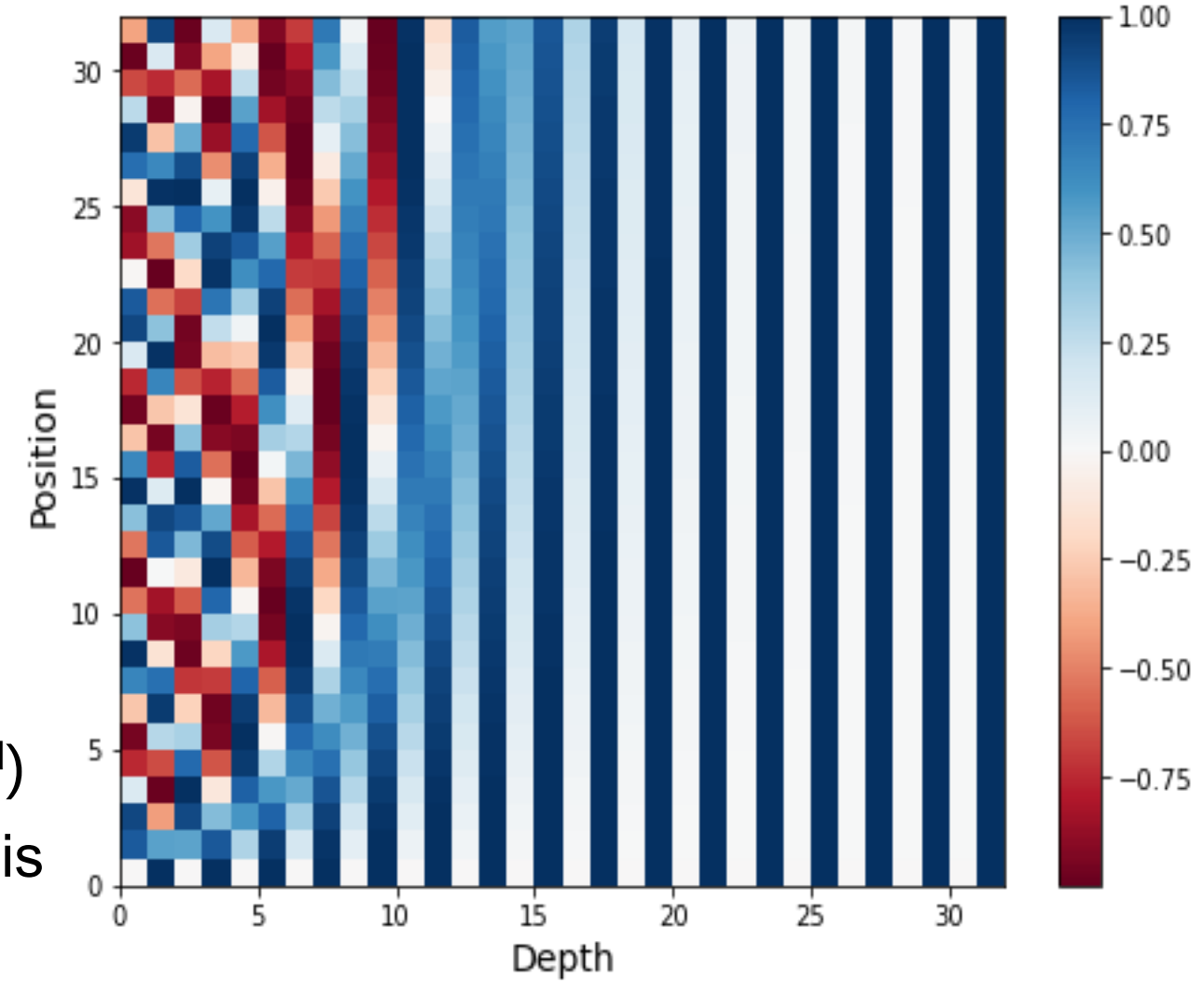
- Preserves information about absolute positioning of tokens in a sequence
- Required to preserve the syntactic and semantic meaning

- Mathematical Formula:

  $PE(pos,2i) = \sin(pos/10000^{2i/d})$

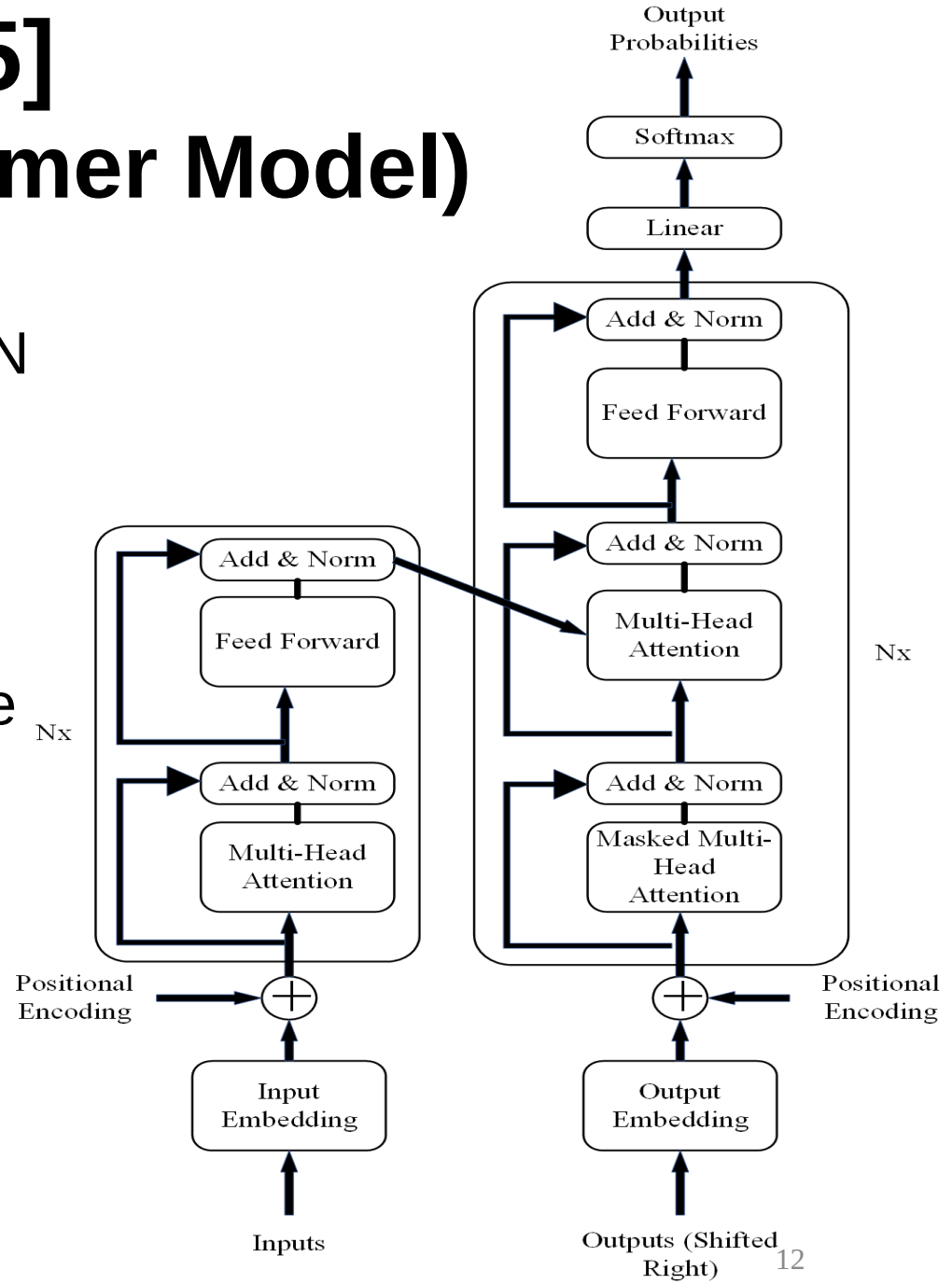
  $PE(pos,2i+1) = \cos(pos/10000^{2i/d})$

  Where ‘pos’ is the position and ‘i’ is the dimension



Methodology - [5] (Encoder–Decoder Transformer Model)

- LSTM solved vanishing gradient problem of RNN
- Transformer uses less training parameters than LSTM
- Allows more parallelization and infinite reference window
- Encoder layers map input sequences and hold learned information
- Decoder layers generate output sequences



Methodology - [6] (Encoder)

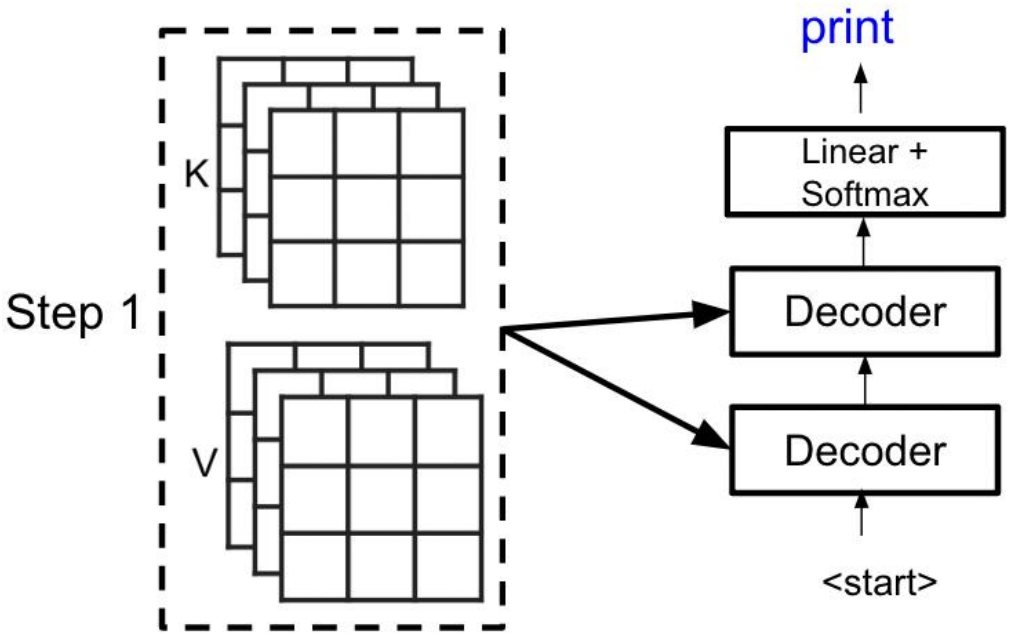
- Preprocessed input multiplied with weight matrices to get Q, K and V matrices.
- Query and a set of Key value pairs mapped to an output
- Attention matrix obtained using the formula,

$$\text{Attention } (Z) = \text{softmax}(QK^T/\sqrt{dk}) \times V$$

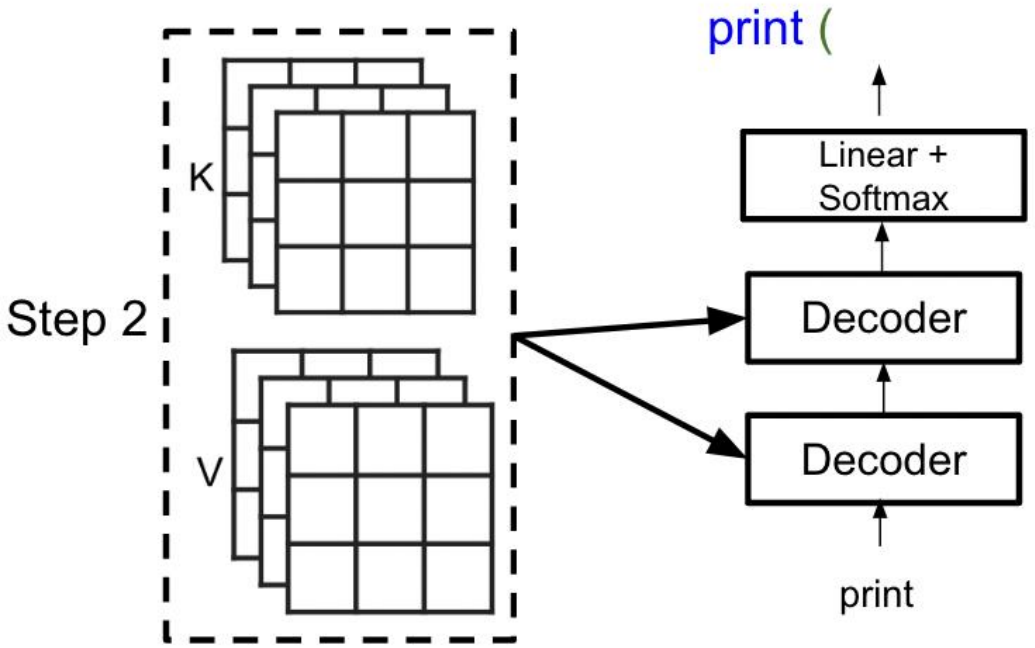
- Pass attention matrices through feed forward layer
- Generated encoder output acts as decoder input

Methodology - [7] (Decoder)

- Autoregressive decoder



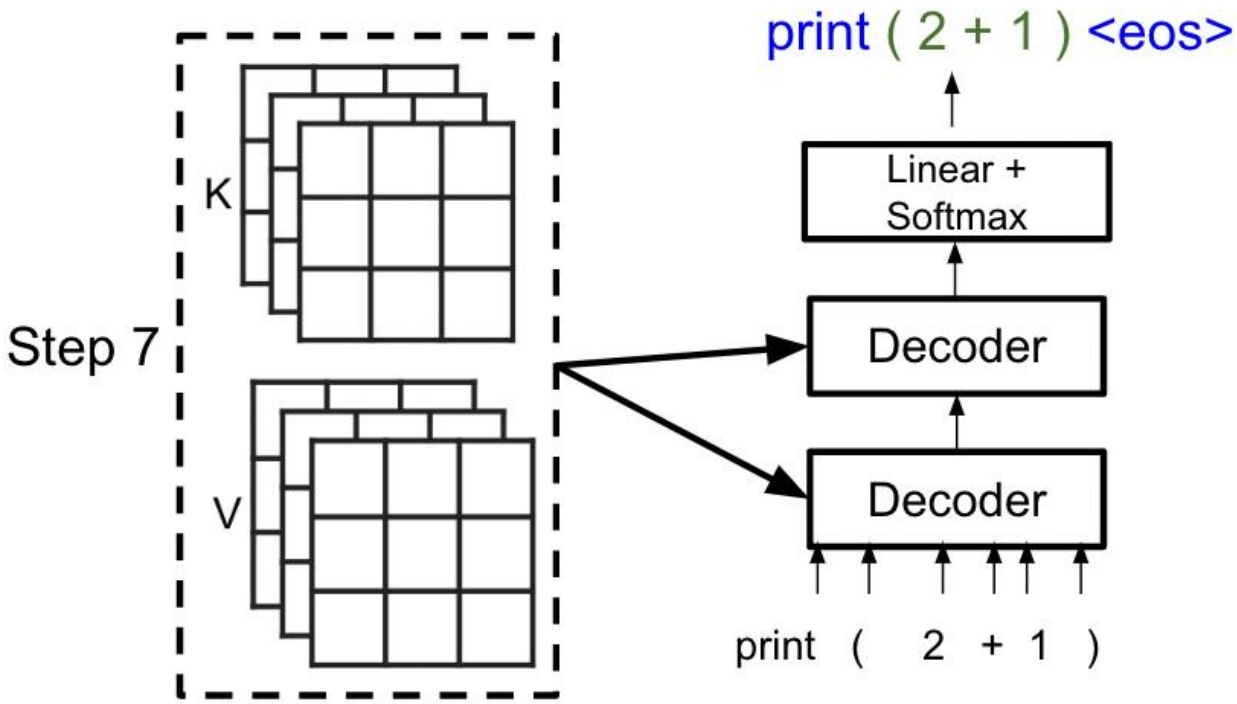
Start token as initial input that generated 'print' as output



'print' as input that generated '(' as output

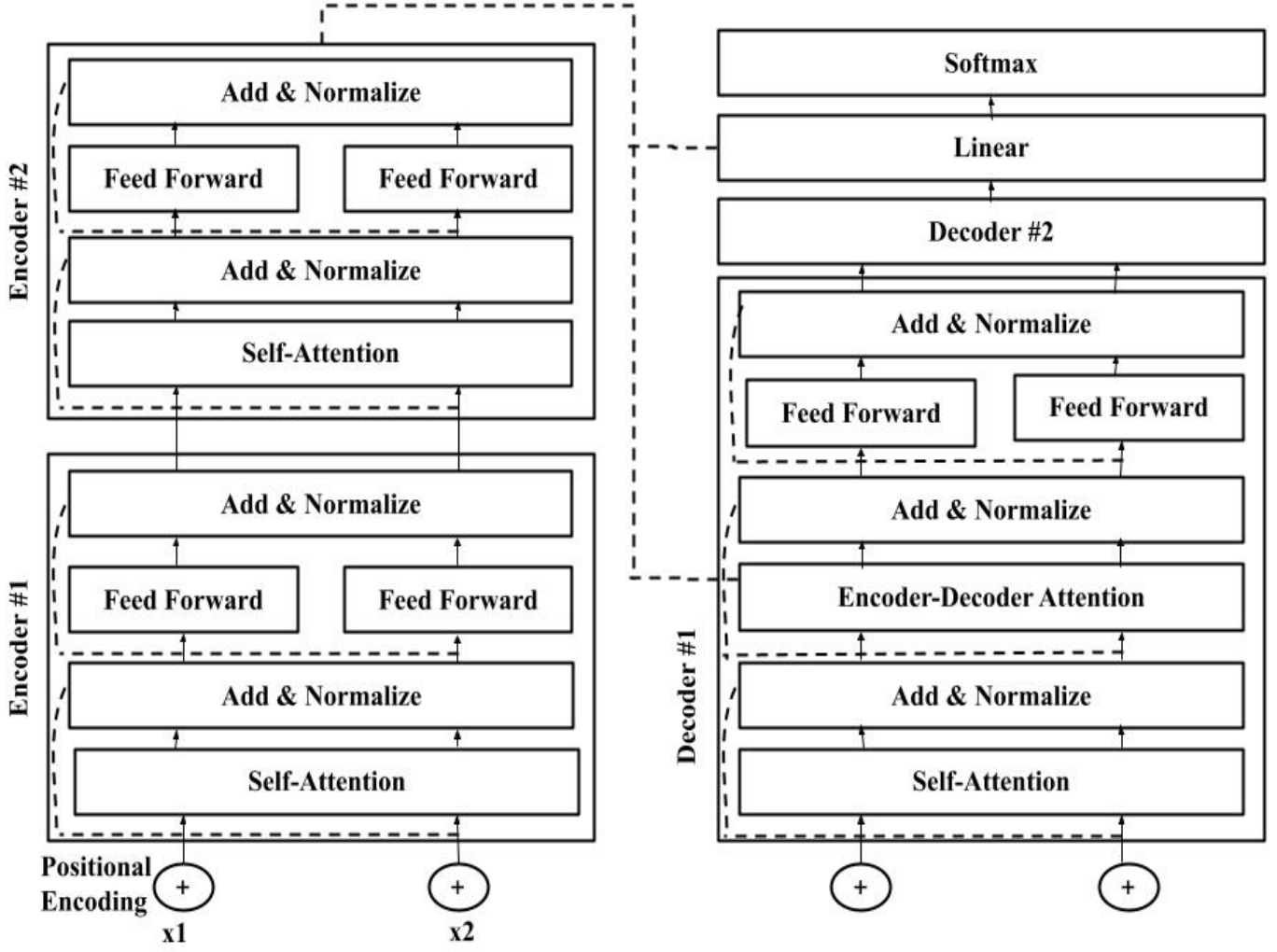
Methodology - [8]
(Decoder – Output, Postprocessing and Concatenation)

- Computational output of decoder in numeric vector form
- Post processing the vector to obtain tokens: ["", 'print', '(', '2', '+', '1', ')’]
- Concatenation to obtain source code: print(2+1)



Methodology - [9] (CodeT5 Architecture)

- T5 stands for Text-to-Text Transfer Transformer
- CodeT5 builds on an encoder-decoder framework
- Same architecture as T5
- Improved bidirectional conversion between NL and PL
- Trained on Python, Java, JavaScript, PHP, Ruby, Go, C, and C#



Methodology - [10]
(Model Evaluation)

- Sparse Categorical Cross Entropy Loss Function
  - Calculates difference between two probability distributions for a set of events
  - Used to evaluate how well the algorithm models the dataset
- BLEU Score
  - Compares generated translation to reference translation
  - Used to evaluate machine-translated text
- Similarity Index
  - Finds exact token to token match between given sequences
  - Calculated using sequence matcher from difflib python library

Methodology - [11]
(Optimizer and Hyperparameters)

- Adam Optimizer
  - Makes use of adaptive moment estimation of the first moment (mean) and the second moment (variance)
  - Combines the advantages of Root Mean Square Propagation and Adaptive Gradient Algorithm

Hyperparameters	Vanilla Model	CodeT5-small
Number of epochs	30	6
Warmup steps	4000	1000
Learning rate (α)	1e-3	8e-4
Exponential decay rate for First Moment (β_1)	0.9	0.9
Exponential decay rate for Second Moment (β_2)	0.98	0.999
Epsilon (ε)	1e-3	1e-8

Methodology - [12]
(Comparison of Model Parameters)

- Model Parameters
  - Vanilla model has around 13 million trainable parameters
  - CodeT5-small has 60.5 million trainable parameters

Parameters	Vanilla Model	CodeT5-small
Size of Encoder-Decoder layer(dmodel)	128	512
Size of Feed forward Layer (dff)	512	2048
Number of Layers	4	6
Attention Heads	8	8

Methodology - [13]
(Hardware and Software Requirements)

- Google Colab
  - Hosted Jupyter notebook service for model training
- GPU
  - Processor to accelerate parallel processing
  - GPU of 12.68 GB RAM used
- TPU
  - ASIC to speed up AI calculations
  - TPU of 30.8 GB RAM
- NLTK
  - Natural Language Translation Toolkit
  - Python library for NLP used for tokenization
- Keras, TensorFlow and PyTorch
  - Python ML framework used for model training
  - Save and load transformer model
- Django Framework
  - Python-based web framework
  - Used for user interface

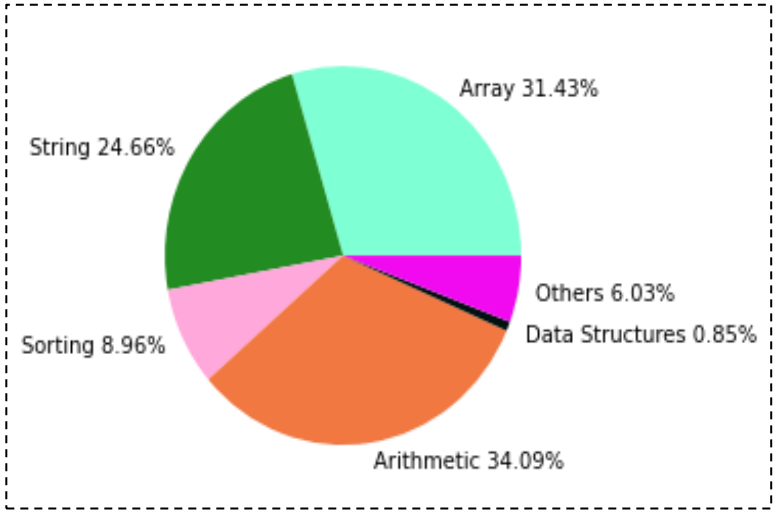
Dataset Exploration - [1]

- SPoC: Search-based Pseudocode to Code
  - 18,356 human authored C++ programs for 677 problems
  - Previously split as training set with 14,546 programs
  - Validation set with 2033 programs
  - Test set with 1777 programs

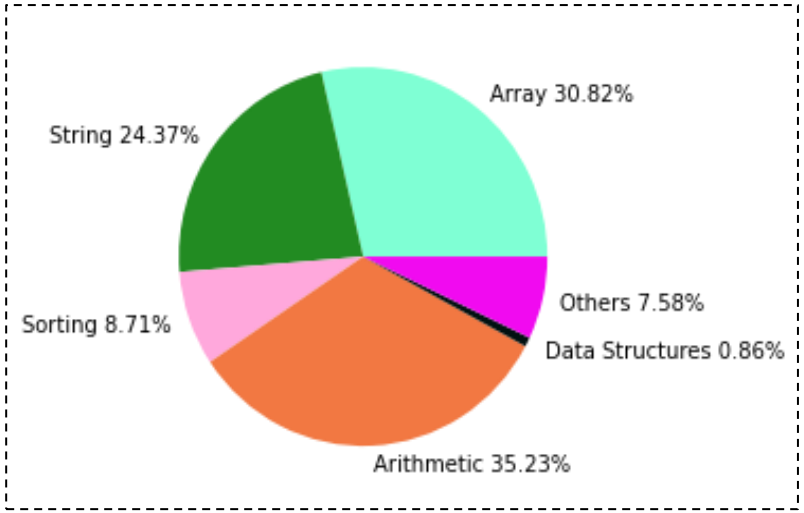
Dataset Exploration - [2] (SPoC Dataset Preview)

text	code	workerid	probid	subid	line	indent
in the function gcd(a,b=integers)	int gcd(int a, int b) {	38	13A	41120785	0	0
if b=1 return a, else call function gcd(b, a%b)	return !b ? a : gcd(b, a % b);	38	13A	41120785	1	1
NaN	}	38	13A	41120785	2	0
NaN	int main() {	38	13A	41120785	3	0
n , nn, ans = integers with ans =0	int n, nn, ans = 0;	38	13A	41120785	4	1
Read n	cin >> n;	38	13A	41120785	5	1
for i=2 to n-1 execute	for (int i = 2; i <= n - 1; ++i) {	38	13A	41120785	6	1

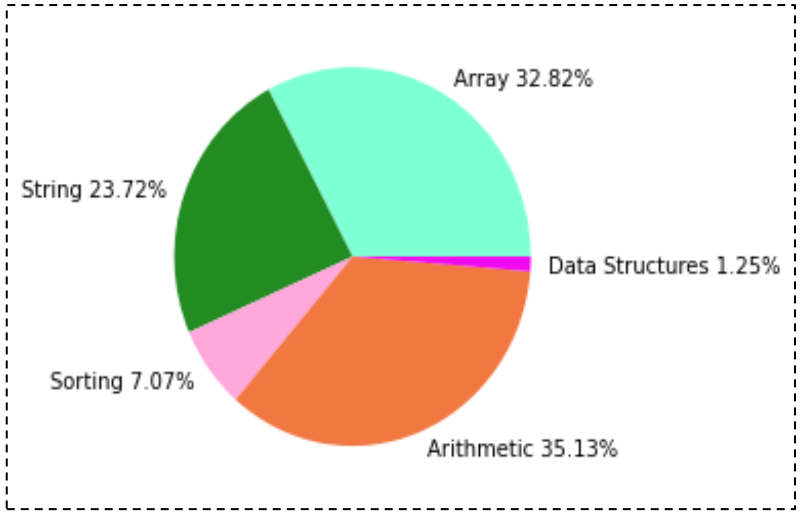
Dataset Exploration - [3]
(Dataset problem categorization)



Training Set



Validation Set

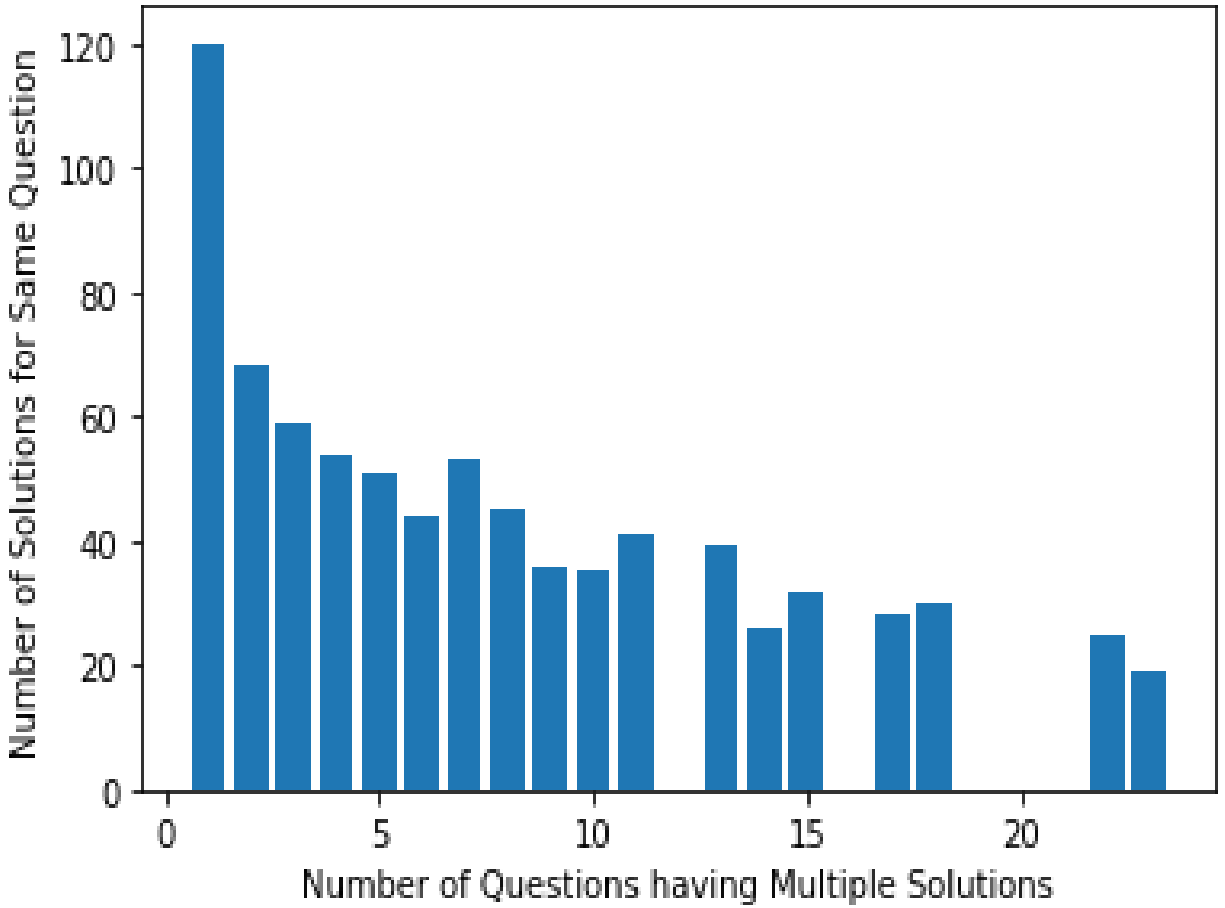


Test Set

- Majority of problems of array and arithmetic operations
- String manipulation problems also prominent
- Very few data structure related problems
- File handling and inheritance grouped as others

Dataset Exploration - [4]
(Question Count with Multiple Solutions)

- Some questions having multiple solutions
- Variations brought changing language of pseudocode
- One question with 120 solutions
- Dataset suspected to be homogeneous



Results- [1]
(Dataset Preprocessed)

- Header file included
- The main() declaration added in records
- NaN values removed from each program

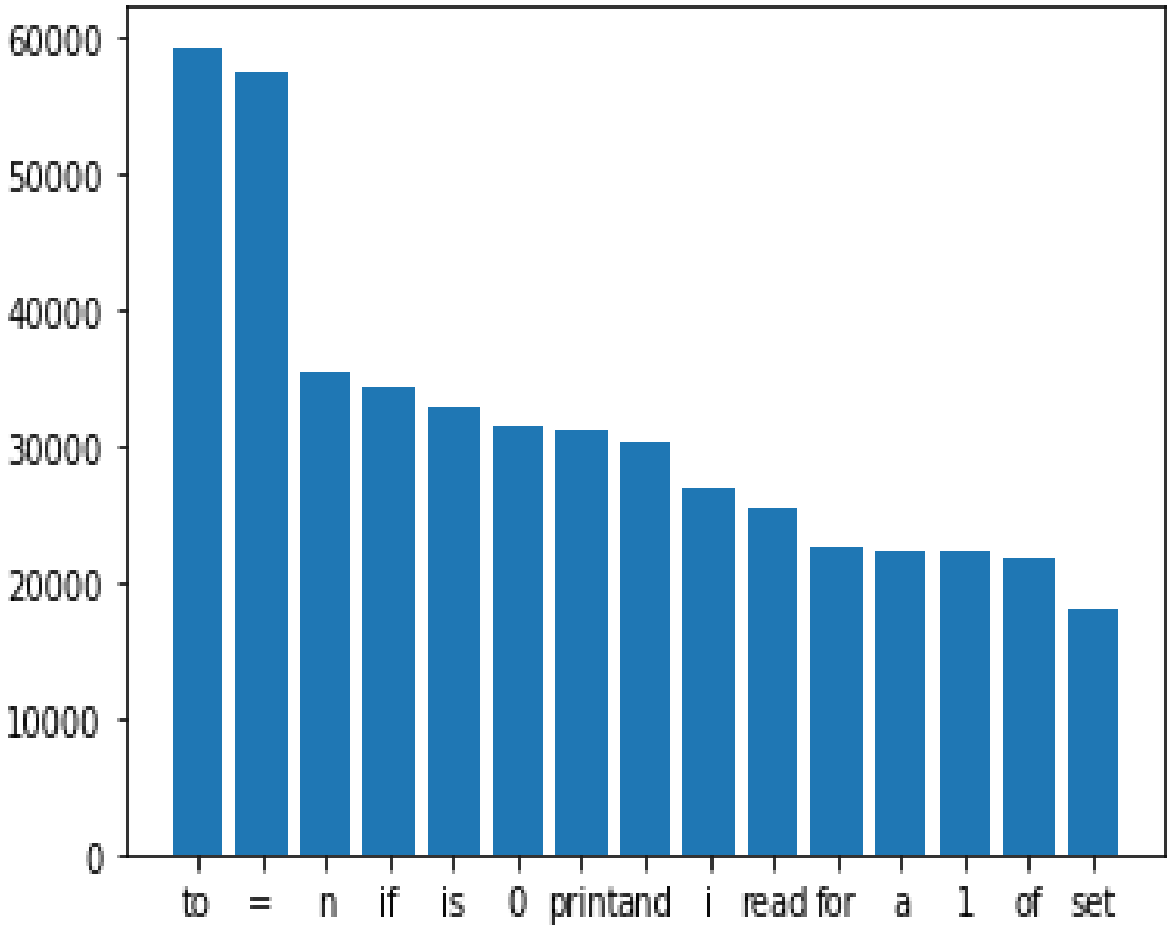
Pseudocode	Program

let a and b be strings
n = integer
st = set of strings
read n
for integer i = 0 to n exclusive
read a and b
insert a + " " + b into st
 	
print size of st and a new line
	#include <iostream>
	using namespace std;
	int main(){
	string a, b;
	int n;
	set<string> st;
	cin >> n;
	for (int i = 0; i < n; i++) {
	cin >> a >> b;
	st.insert(a + " " + b);
	}
	cout << st.size() << endl;
	return 0;
	}

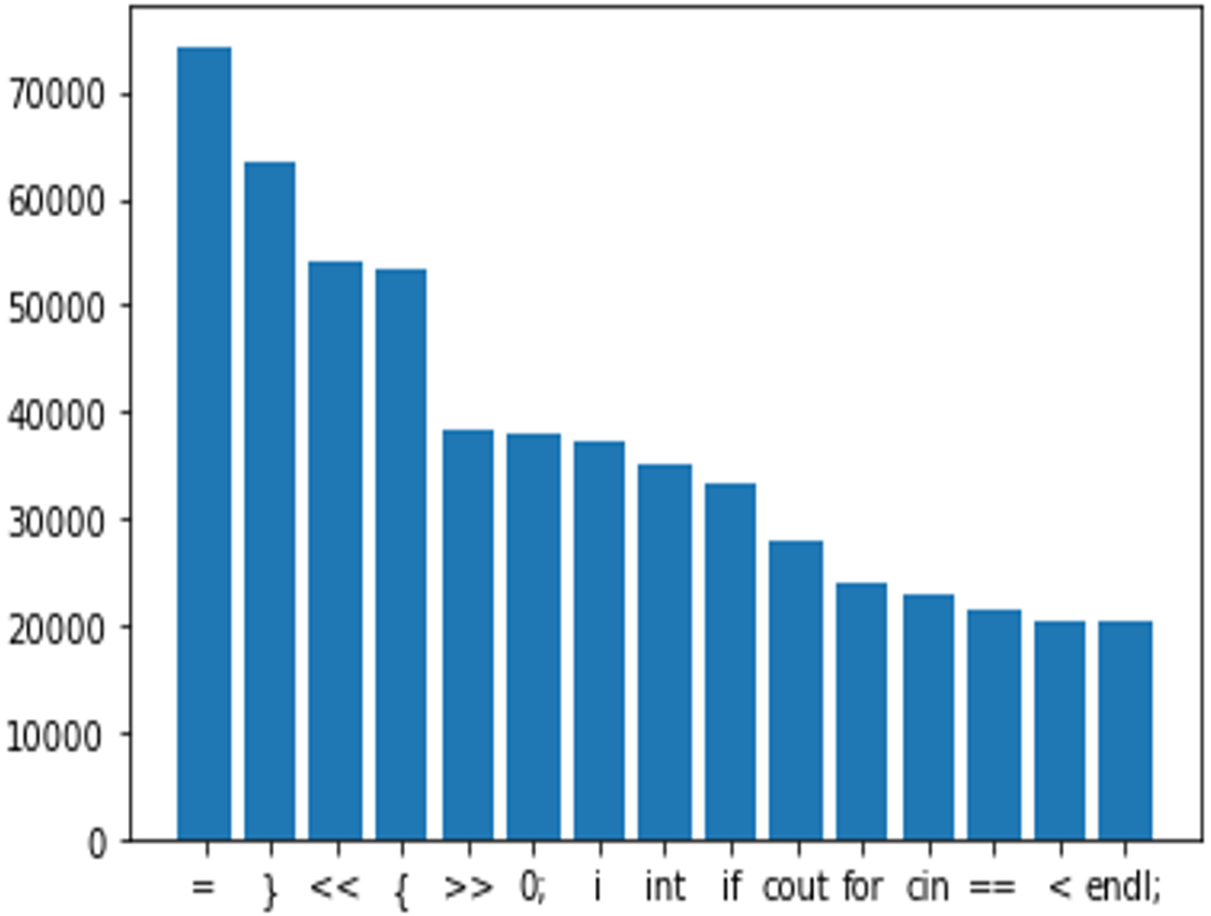
Results- [2]

(Top 15 Tokens from Training and Validation sets)

- Pseudocode Tokenization

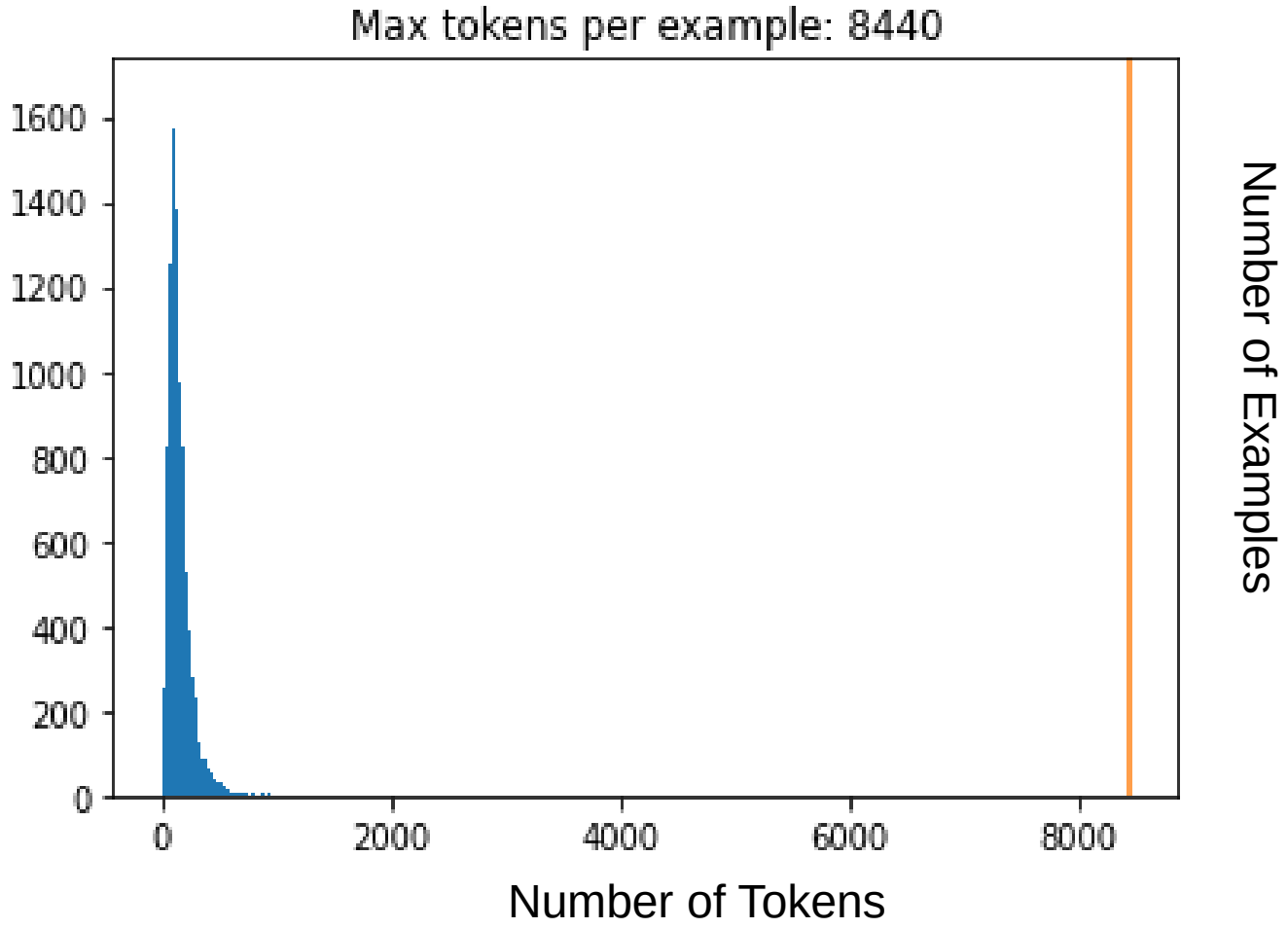


- Program Tokenization



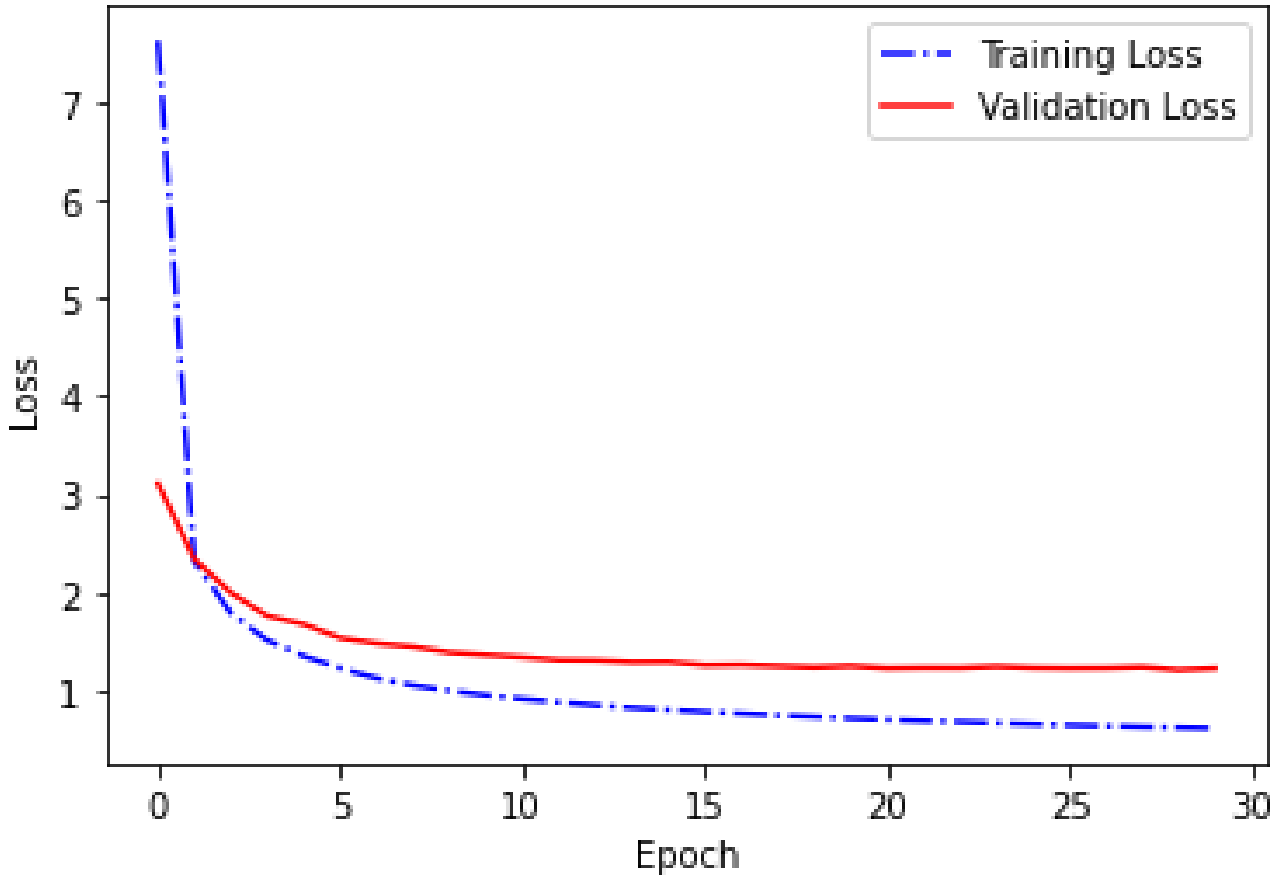
Results- [3] (Tokenization)

- 2285 unique tokens in pseudocode
- 1989 unique tokens C++ code vocabulary
- Most C++ code with tokens less than 1000



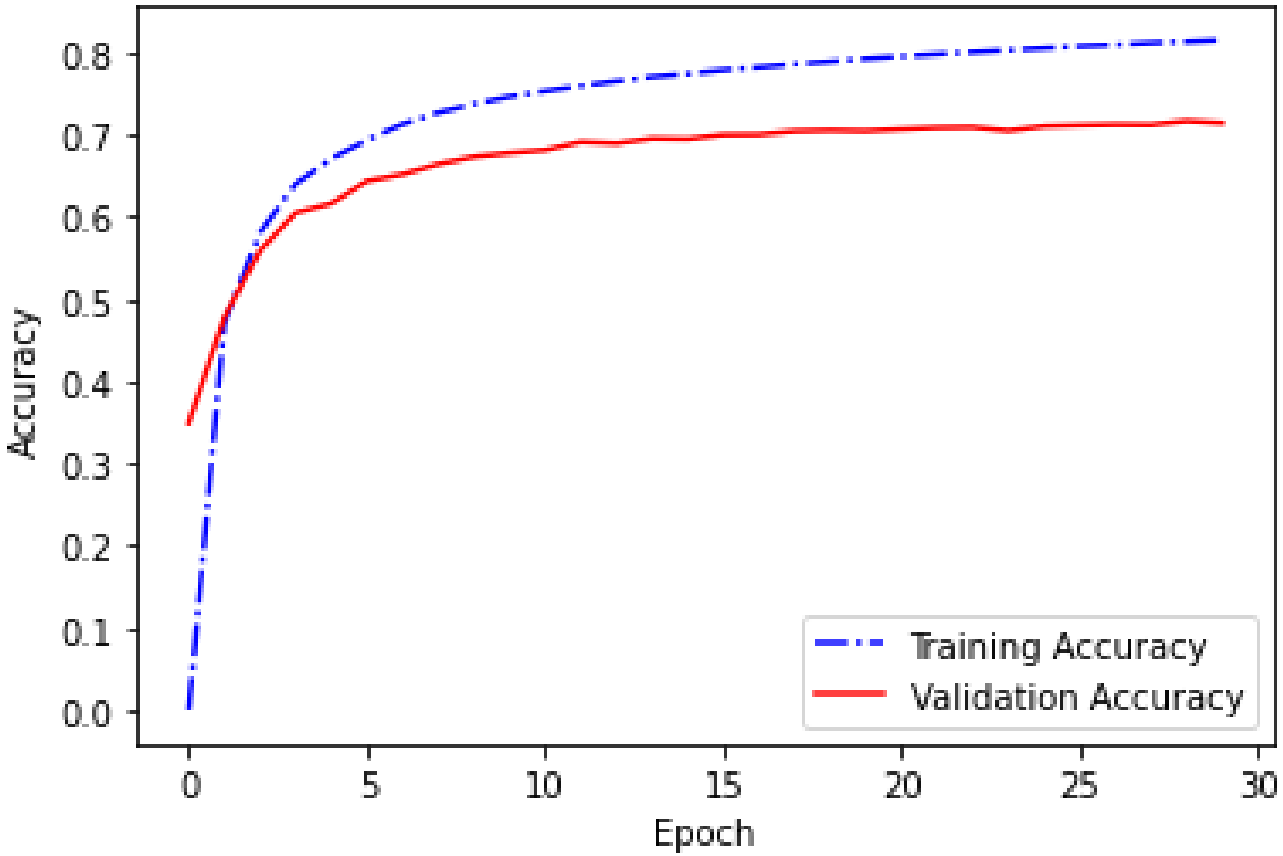
Results- Vanilla Transformer Model [4] (Training and Validation Loss)

- Transformer model trained for 30 epochs
- Training loss decreased from 7.3541 to 0.6095
- Validation loss decreased from 2.7669 to 0.772



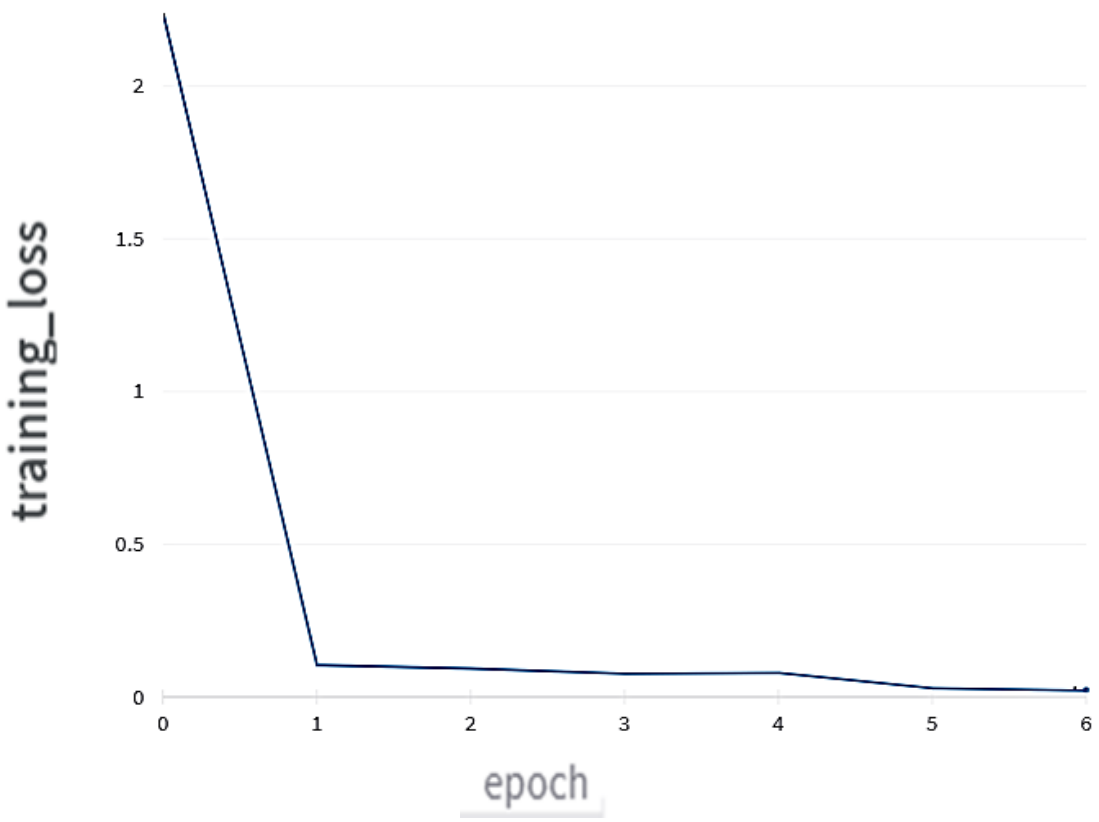
Results- Vanilla Transformer Model [5] (Training and Validation Accuracy)

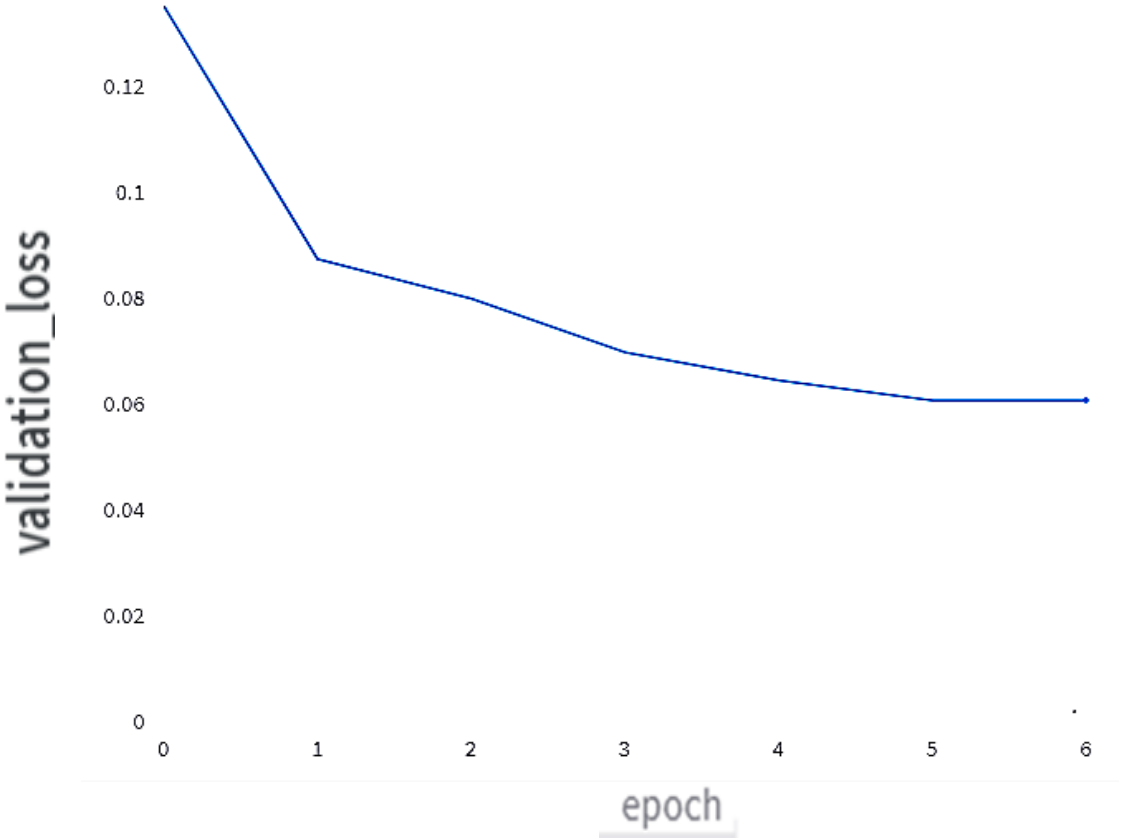
- Training accuracy was achieved up to 81.5%
- Validation accuracy was achieved up to 79.9%



Results- CodeT5 Model [6] (Training and Validation Loss)

- Model trained for 6 epochs





Training loss decreased from 2.239 to 0.02313

Validation loss decreased from 0.1354 to 0.06073

Results- [7]
(Model Output : Print String)

Input	Output by Vanilla transformer model	Output by CodeT5 model

Print
"This is sentence"
	# include < iostream >
	using namespace std ;
	int main ( ) {
	cout < < " karen " < < endl
	; return 0 ; }
		#include <iostream>
		using namespace std;
		int main(){
		cout << "This is sentence"
		<< endl;
		return 0;
		}

	Similarity Index = 0.82 BLEU Score = 0.32	Similarity Index = 0.9 BLEU Score = 0.64

Results- [8]
(Model Output Example)

Input	Output by Vanilla transformer model	Output by CodeT5 model

declare integer a
input a
print a
	# include < iostream >
	using namespace std ;
	int main ( ) {
	int a ;
	cin >> a ;
	cout << a << endl ;
	return 0 ; }
		#include <iostream>
		using namespace std;
		int main() {
		int a;
		cin >> a;
		cout << a << endl;
		return 0 ;
		}

	Similarity Index = 0.88 BLEU Score = 0.65	Similarity Index = 0.93 BLEU Score = 0.85

Results- [9]
(Model Output : Fibonacci)

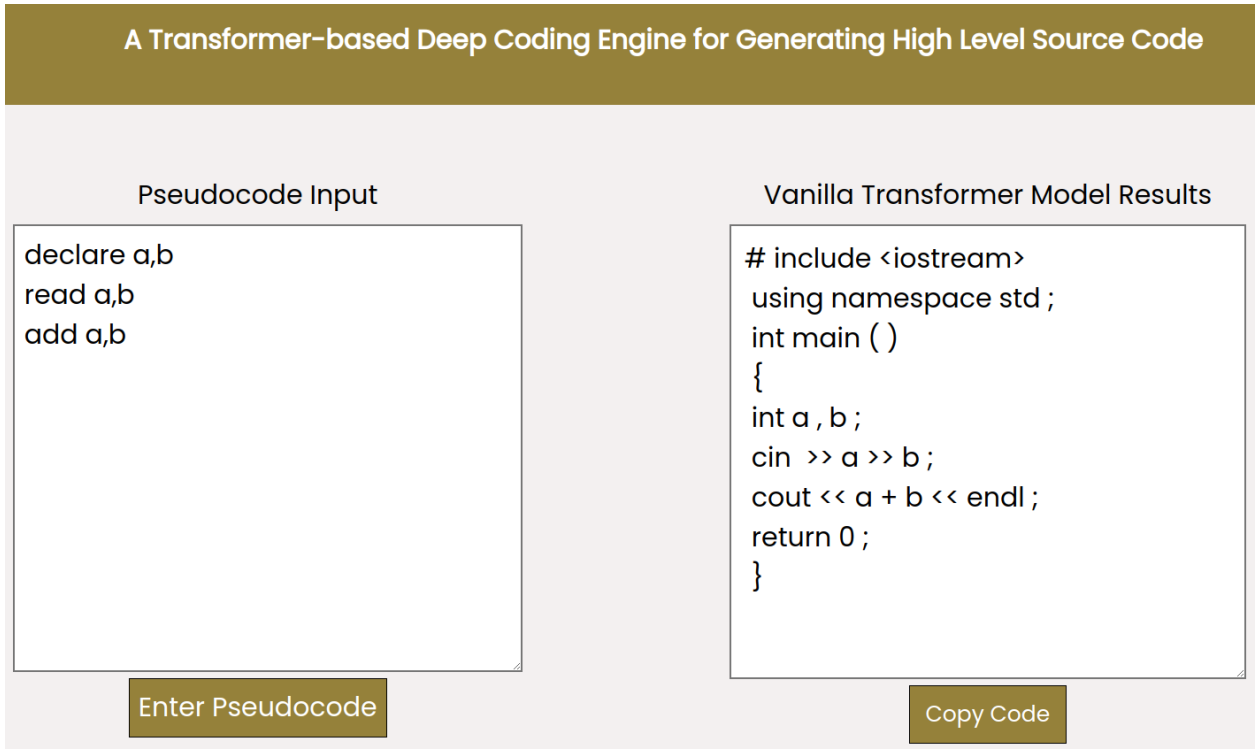
Input	Output by Vanilla transformer model	Output by CodeT5 model

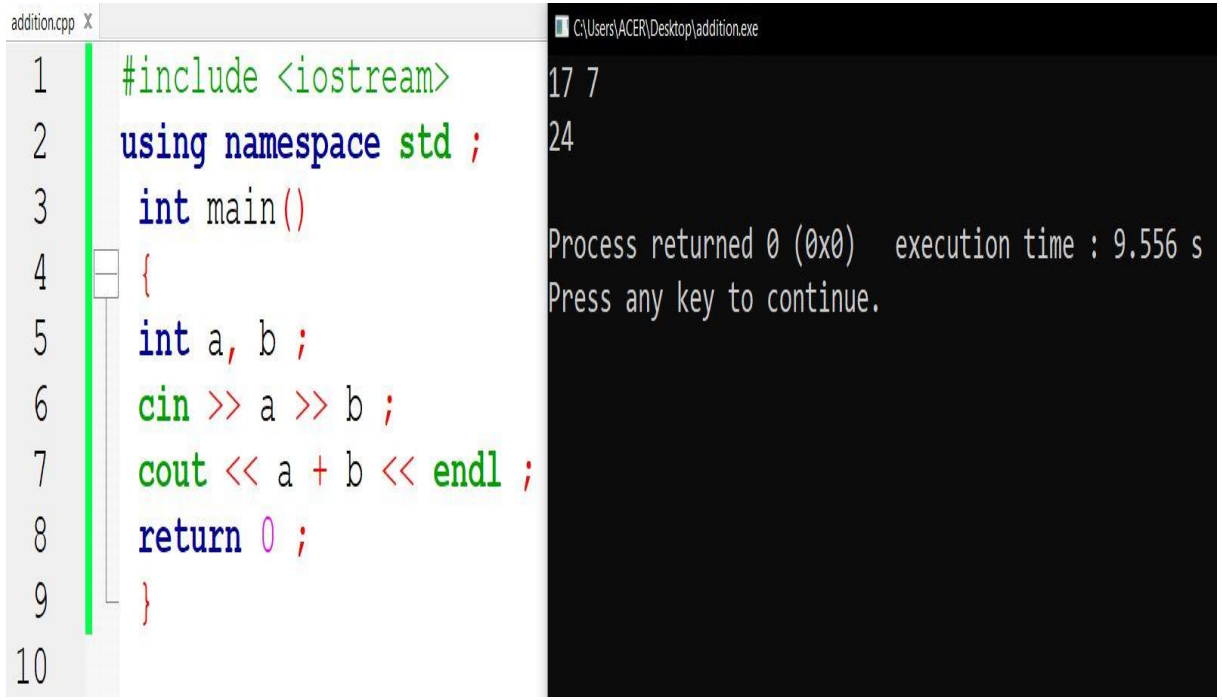
Declare variable n1, n2, sum, n, i
n1 = 0
n2 = 1
Read n
print n1
print n2
for i = 2 to n :
  sum = n1 + n2
  print sum
  n1 = n2
  n2 = sum
	include < iostream >
	using namespace std ;
	 int main ( ) {
	 int n1 , n2 , n , sum = 0 ;
	 cin > > n1 > > n2 > > n ;
	for (int i = 1 ; i < = n1 ; i + +) {
	n1 = n2 ;
	 n2 = n1 ;
	 }
	 n1 = n2 ;
	 n2 = n2 ;
	 cout < < n1 < < endl ;
	 return 0 ; }
		#include <iostream>
		using namespace std;
		int main() {
		int n1, n2, sum, n, i;
		n1 = 0;
		n2 = 1;
		cin >> n;
		cout << n1 << endl;
		cout << n2 << endl;
		for (i = 2; i <= n; i++) {
		sum = n1 + n2;
		cout << sum << endl;
		  n1 = n2;
		  n2 = sum;
		}
		return 0;}

	Similarity Index = 0.33 BLEU Score = 0.3	Similarity Index = 0.66 BLEU Score = 0.49

Results- [10]
(User Interface)

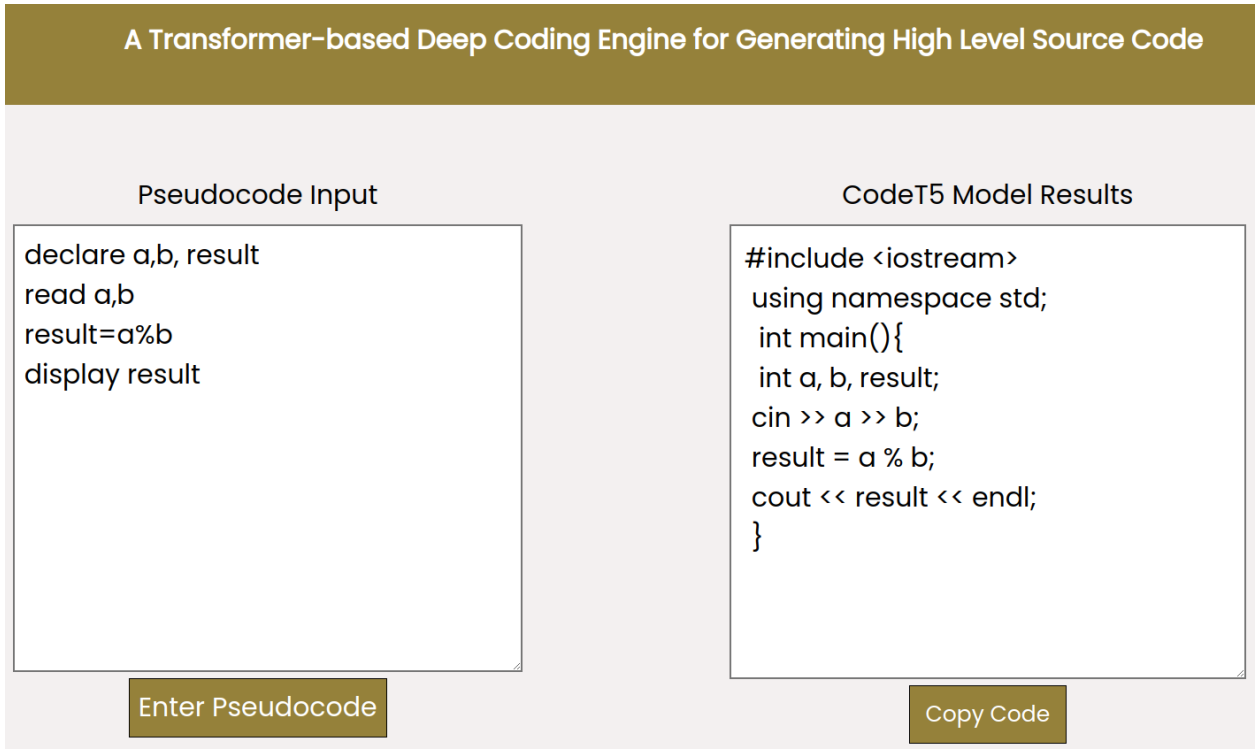
- Addition of two numbers using Vanilla model

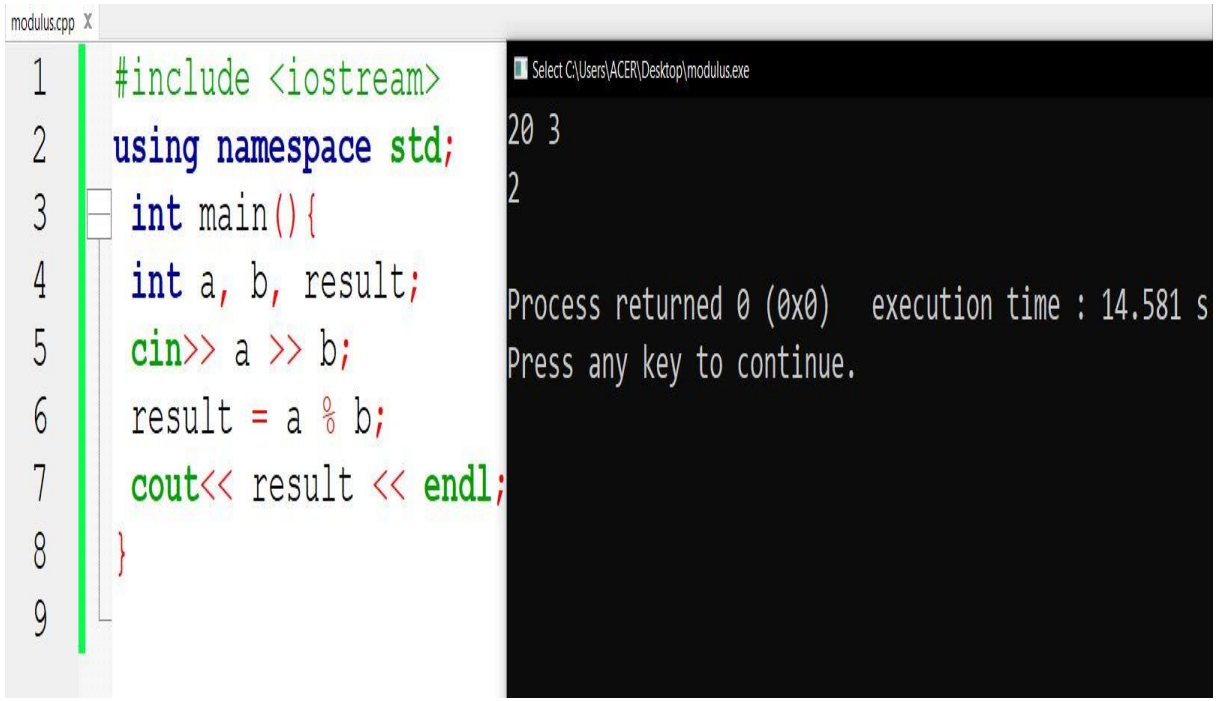




Results- [11] (User Interface)

- Modulus of two numbers using CodeT5 model





Discussion and Analysis - [1] (Model Performance)

- CodeT5 model had better performance than vanilla transformer
- Model performance evaluated using BLEU score and similarity index on test set.

Models	Dataset	Mean Similarity Index	Mean BLEU Score
Transformer-base	SPoC	16.11%	22.77%
CodeT5-small		**87.99%**	**84.91%**

Discussion and Analysis - [2] (Problem Complexity)

- For better results, pseudocode needs to be very specific
- Needs clear mention of the data types, operators and variables
- Needs declarations of intermediate steps for logical operations
- Model performed well for array, string, arithmetic and logical operations
- Model may not perform well for programs related to trees, heaps, linked list, pointers, graphs

Future Enhancements

- Transfer learning on larger pre-trained transformer models
- Increasing the data in dataset used for training
- Generating pseudocode to code in multiple programming languages
- Performing problem description to code generation
- Fine-tuning hyperparameters using Grid search
- Replacing BLEU score with CodeBLEU score

Conclusion

- Explored base transformer as well as pretrained CodeT5 model
- Obtained results for transformer model trained from scratch
- Performed transfer learning on CodeT5 transformer model
- Problem domains like arithmetic, string, sorting, array tested
- Minimize the time and effort needed to develop code
- Results fulfilled defined objectives

References - [1]

[1] S. Kulal et al., “SPoC: Search-based Pseudocode to Code”, Advances in Neural Information Processing Systems, vol. 32,Jun. 2019, doi: 10.48550/arxiv.1906.04908

[2] “CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation” [Online]. Available: https://arxiv.org/pdf/2109.00859.pdf [Accessed Feb. 7, 2023].

[3] “Transformer Architecture: The Positional Encoding – Amirhossein Kazemnejad’s Blog.” [Online]. Available: https://kazemnejad.com/blog/transformer_architecture_positional_encod in g/ [Accessed Jun. 25, 2022].

[4] “BERT - Tokenization and Encoding | Albert Au Yeung.” [Online]. Available: https://albertauyeung.github.io/2020/06/19/bert-tokenization.html/ [Accessed Mar. 04, 2023].

References - [2]

[5] “Understanding Encoder-Decoder Sequence to Sequence Model” Understanding Encoder-Decoder Sequence to Sequence Model | by Simeon Kostadinov | Towards Data Science (accessed Jun. 24, 2022).

[6] “Illustrated Guide to Transformers- Step by Step Explanation.” Illustrated Guide to Transformers- Step by Step Explanation | by Michael Phi | Towards Data Science (accessed Jun. 24, 2022).

[7] “Natural Language Processing — Beginner to Advanced (Part-2) | by Ishan Singh | Towards Data Science.” [Online]. Available: https://towardsdatascience.com/lexical-processing-for-nlp-basic-9fd9b7529d06 [Accessed Jun. 25, 2022].

[8] “CodeBLEU: a Method for Automatic Evaluation of Code Synthesis”. [Online]. Available: https://arxiv.org/pdf/2009.10297.pdf [Accessed Feb. 7, 2023].