



Cairo University

TRANSFORMER-BASED MODEL FOR COMPUTER CODE GENERATION TO ASSIST PROGRAMMERS

By

Ahmed Shokry Mahmoud Soliman

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Computer Engineering

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2024

TRANSFORMER-BASED MODEL FOR COMPUTER CODE GENERATION TO ASSIST PROGRAMMERS

By

Ahmed Shokry Mahmoud Soliman

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Computer Engineering

Under the Supervision of

Prof. Dr. Samir Shaheen Dr. Mayada Hadhoud

Professor

Associate Professor

Computer Engineering

Department of Computer Engineering

Faculty of Engineering, Cairo University Faculty of Engineering, Cairo University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2024

TRANSFORMER-BASED MODEL FOR COMPUTER CODE GENERATION TO ASSIST PROGRAMMERS

By

Ahmed Shokry Mahmoud Soliman

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Computer Engineering

Approved by the Examining Committee:

Prof. Dr. Samir Shaheen,

Thesis Main Advisor

Dr. Mayada Hadhoud,

Thesis Advisor

Prof. Dr. Reda Abdelwahab,

Internal Examiner

Faculty of Computers & Artificial Intelligence, Cairo University

Prof. Dr. Mohamed Zaki,

External Examiner

Faculty of Engineering, Al-Azhar University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY

GIZA, EGYPT

2024

Engineer: Ahmed Shokry Mahmoud Soliman
Date of Birth: 08/08/1990
Nationality: Egyptian
E-mail: ahmed.shokry.soliman@gmail.com
Phone.: +201064666545
Address: Toukh – Kalyobia – Egypt
Degree: Master of Science
Department: Computer Engineering
Supervisors:
Prof. Dr. Samir Ibrahim Mohamed Shaheen
Dr. Mayada Mansour Ali Hadhoud



Examiners:
Prof. Dr. Samir Shaheen (Thesis main advisor)
Dr. Mayada Hadhoud (Advisor)
Prof. Dr. Reda Abdelwahab (Internal examiner)
Faculty of Computers and Artificial Intelligence, Cairo University
Prof. Dr. Mohamed Zaki (External examiner)
Department of Computer Engineering, Al-Azhar University

Title of Thesis:

Transformer-Based Model For Computer Code Generation To Assist Programmers

Keywords:

Code Assistance; Code Generation; Transformer-Based Processing; MarianCG; Language Models

Summary:

Code generation is the task of automatically generating and constructing code from a natural language description, and it has gained significant attention in recent years due to the increasing popularity of large language models (LLMs). In this thesis, new models are proposed to solve the code generation problem, including MarianCG which is fine-tuned on a machine translation model, hybrid models with pre-trained language models as encoders and Marian Neural Machine Translation Decoder, and a LLM for code generation using Llama-2 model with PEFT and QLoRA techniques. The results of these experiments were also analyzed to display error/warning messages and refine the generated codes.

Disclaimer

I hereby declare that this thesis is my own original work and that no part of it has been submitted for a degree qualification at any other university or institute.

I further declare that I have appropriately acknowledged all sources used and have cited them in the references section.

Name: Ahmed Shokry Mahmoud Soliman

Date:

Signature:

Dedication

This research work is heartily and proudly dedicated to the people who serve as an inspiration through my journey in developing this work. From my family including my wife, my father, my mother, and my brothers and guardians, to my supervisors who put me on the road of how to improve my skills, and employ these skills to have a great output through some efforts. Thanks to my supervisors including Prof. Samir Shaheen and Dr. Mayada Hadhoud, and all of the support from Prof. Mohamed Zaki. Also, inspiration by my classmates and a circle of friends who extended their help in the midst of problems while doing this work.

To the faculty of Engineering and staff of Computer Engineering at Cairo University in Egypt.

Above all, to our God Almighty Who provided us His blessings in our everyday lives, especially for the strength, courage, patience, wisdom, time, and guidance in the realization of this work.

Acknowledgements

Many thanks to my family: my wife, father, mother, and brothers for their support. I would like to express my gratitude to my supervisors in this work Prof. Samir Shaheen and Dr. Mayada Hadhoud, and many thanks to Prof. Mohamed Zaki, who guided me a lot throughout my Master's work. I would also like to thank my friends who supported me and offered deep insight into the study.

Table of Contents

Disclaimer	i
Dedication	ii
Acknowledgements	iii
Table of Contents	iv
List of Tables	viii
List of Figures	ix
List of Symbols and Abbreviations	xii
List of Publications	xiii
Abstract	xiv
1 INTRODUCTION	1
1.1 Motivation	3
1.2 Proposed Solution	4
1.3 Thesis Contribution	6
1.4 Thesis outline	7
2 BACKGROUND	8
2.1 Deep Learning and Transformer-Based Processing	8
2.1.1 Deep Learning	9
2.1.2 Transformer-Based Processing	9
2.1.3 Deep Learning Techniques for Language Representation	10
2.2 Language Modeling.....	12
2.2.1 Statistical language models	13
2.2.2 Neural Language Models(NLM)	14
2.3 Sequence Models.....	15
2.3.1 Recurrent Neural Network (RNN)	15

2.3.2	LSTM	16
2.3.3	Transformers	17
2.3.4	The evolution of language models	19
2.4	Transformer-Based Processing	21
2.5	Transfer Learning and Knowledge Distillation	22
2.5.1	Pre-trained Language Models	22
2.5.2	Transfer Learning	23
2.5.3	Knowledge Distillation	23
2.6	Evaluation metrics of seq2seq models	25
2.6.1	BLEU Score	26
2.6.2	Exact Match Accuracy	28
2.6.3	ROUGE Score	28
2.7	Error Correction and Code Enhancement	30
2.7.1	Linting and Error Analysis	30
2.7.2	Code Enhancement and Reformatting	31
3	LITERATURE REVIEW	33
3.1	Code Generation Techniques.....	34
3.1.1	Semantic Parsing Based Techniques	35
3.1.2	Tree-based Techniques	39
3.1.3	Deep Learning based Techniques.....	41
3.2	Code Generation Datasets.....	45
3.2.1	CoNaLa Dataset.....	45
3.2.2	DJANGO Dataset.....	46
3.2.3	CodeSearchNet Dataset	47
4	PROPOSED APPROACH	49
4.1	Introduction	49
4.2	MarianCG Model.....	50
4.2.1	MarianCG Tokenization.....	53
4.2.2	MarianCG Embedding and Positional Embedding	53
4.2.3	Marian Encoder and Decoder Architecture	54
4.3	Proposed Hybrid Models.....	56

4.3.1	RoBERTaMarian Model.....	57
4.3.1.1	RoBERTa Poller Layer.....	59
4.3.2	BERTMarian Model	61
4.3.3	ELECTRAMarian Model.....	62
4.3.4	LUKE Marian Model	64
4.3.4.1	LUKE Pooler Layer.....	66
4.4	Error Analysis and Correction.....	68
4.4.1	Error Analysis in the Generated Code.....	68
4.4.2	Code Refining and Correction.....	69
4.5	Summary on the Proposed Approaches.....	71
4.5.1	Tokenization and Transfer Learning Insights	72
4.5.2	Comparative Analysis.....	73
5	EXPERIMENTAL RESULTS	75
5.1	Datasets	75
5.2	Experimental setup.....	77
5.3	Experimental Results.....	77
5.3.1	MarianCG Model	78
5.3.1.1	Experiment 1.....	78
5.3.1.2	Experiment 2.....	78
5.3.2	RoBERTaMarian Model.....	82
5.3.3	BERTMarian Model	84
5.3.4	ELECTRAMarian Model.....	86
5.3.5	LUKE Marian Model	87
5.4	Discussion	89
5.5	Error/Warning Analysis and Refining.....	94
6	MULTILINE CODE GENERATION WITH LLM	100
6.1	Introduction	100
6.2	Fine-tuning Large Language Models(LLMs)	102
6.3	PEFT, LoRA and QLoRa	103
6.4	Llama2	104
6.5	Our Proposed Approach: Llama2-CodeGen	106

6.6	Llama2-CodeGen Example.....	108
6.7	Prompt Engineering And Tuning, Prefix Tuning, And Adapter Tuning	108
6.7.1	Prompt Tuning And Prefix Tuning.....	108
6.7.2	Adapter Tuning	109
7	CONCLUSION AND FUTURE WORK	111
7.1	Conclusion.....	111
7.2	Future Work	112
	References	115
	Appendix A TRAINING AND TESTING NOTEBOOKS	130
A.1	GitHub Repositories.....	131
A.2	Notebooks	131
	Appendix B MODELS AND DATASETS	133
B.1	Models	133
B.2	Datasets.....	134

List of Tables

2.1	BLEU score	27
2.2	Python Linters and Code Analysis Tools.....	32
3.1	CoNaLa Dataset NL-Code pairing examples	45
3.2	Code generation examples in the DJANGO dataset.....	46
4.1	Our proposed Code Generation Models	58
4.2	Comparison of Proposed Encoder Models	74
5.1	Datasets in each experiment and distribution of the data	75
5.2	Results of MarianCG model on DJANGO dataset	78
5.3	Results of the first experiment on CoNaLa	79
5.4	Results of MarianCG model on the CoNaLa dataset.....	81
5.5	Performance metrics of all models on the CoNaLa Dataset	90
5.6	Performance metrics of all models on the DJANGO Dataset.....	91
5.7	State of the art Code Generation models on CoNaLa.....	92
5.8	State-of-the-art Code Generation models on DJANGO.....	93
5.9	Error And Warning Analysis on The MarianCG Model	95
5.10	Error And Warning Analysis Through RoBERTaMarian Model	96
5.11	Error And Warning Analysis After Refining and Correction on MarianCG Model.....	97
5.12	Error And Warning Analysis After Refining and Correction on RoBERTaMarian Model.....	99

List of Figures

1.1	Example of Semantic Parsing	1
1.2	Example of Code Generation	2
1.3	Pipelines for the Code Assistance System	5
2.1	Deep Neural Network Architecture [40]	9
2.2	Recurrent Neural Network(RNN) [97]	15
2.3	Feedforward Neural Network and Recurrent Neural Network.....	16
2.4	LSTM [40].....	16
2.5	Transformer Architecture [30]	18
2.6	Attention Mechanism - (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel [30]	19
2.7	The architecture of embeddings from language models (ELMO)-based BiLSTM-CRF model [108].....	20
2.8	T5 Model capabilities [110].....	20
2.9	Transformer timeline and Pre-trained language models . Colors de- scribe Transformer family [115]	23
2.10	Transfer learning and fine-tuning pre-trained models [127]	24
2.11	Fine-tuning pre-trained BERT language model [60]	24
2.12	Knowledge Distillation [129]	25
2.13	Various relationships among teacher-student networks [129].....	25
2.14	The proper pipeline for computing reported matching scores.....	26
2.15	Two examples for measuring the Exact Match Accuracy of two sentences	29
2.16	Example for calculating ROUGE-L.....	29
3.1	The CoNaLa dataset contents	45
4.1	Code generation seq2seq model.....	49
4.2	Marian Model	50
4.3	MarianCG model for Code Generation	51
4.4	MarianCG Model Architecture.....	52

4.5	Preprocessing phase for the data	52
4.6	MarianCG Encoder Decoder representation for code generation	53
4.7	Marian Encoder Architecture	55
4.8	Marian Decoder Architecture.....	55
4.9	Leveraging Pre-trained Language models for building Code Generation Models	57
4.10	RoBERTaMarian Code Generation Model	58
4.11	RoBERTaMarian Model Architecture	59
4.12	RoBERTaMarian Encoder	60
4.13	DistilBERT Model Architecture[200]	61
4.14	BERTMarian Code Generation Model.....	62
4.15	BERTMarian Model Architecture.....	62
4.16	BERTMarian Encoder.....	63
4.17	ELECTRA model like GAN[201]	63
4.18	ELECTRAMarian Code Generation Model	64
4.19	ELECTRAMarian Model Architecture	64
4.20	ELECTRAMarian Encoder	65
4.21	LUKE outputs are contextualized representations for each word and entity[202].....	65
4.22	LUKEMarian Code Generation Model	66
4.23	LUKEMarian Model Architecture.....	66
4.24	LUKEMarian Encoder	67
4.25	AI Assistant Code Process	70
5.1	CoNaLa Dataset(Training-Validation-Testing)	76
5.2	DJANGO Dataset(Training-Validation-Testing)	76
5.3	CoNaLa Dataset(Training-Validation-Testing)	76
5.4	Results on DJANGO	79
5.5	Results on CoNaLa.....	82
5.6	Results and Performance Metrics of our proposed models on the CoNaLa Dataset	91
5.7	Results and performance metrics of our proposed models on the DJANGO dataset.....	92

5.8	All Results of the state of the art models in the code generation problem with CoNaLa	93
5.9	All results of the state-of-the-art models in the code generation on DJANGO with respect to Accuracy	94
5.10	Error Analysis in the MarianCG Model Generated Code	95
5.11	Error Analysis in the Generated Code using RoBERTaMarian Model	96
5.12	Refining and Correction of the Errors in the MarianCG Generated Code	98
5.13	Refining and Correction of the Errors in the RoBERTaMarian Generated Code	98
6.1	Process from fine-tuning an adapter to model deployment	102
6.2	Different finetuning methods and their memory requirements. QLORA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.[208]	104
6.3	LoRA Weights in the fine-tuning using LoRA and PEFT[214].....	105
6.4	Regular Weights in the fine-tuning process[214]	106
6.5	Data Preprocessing and Phases to Configure the Dataset.....	107
6.6	Fine-tuning Llama-2 with Prompt Tuning, Adapter Tuning and LoRA[215]	107
6.7	Prompt Tuning[215].....	109
6.8	Llama-Adapter[215].	110

List of Symbols and Abbreviations

Abbreviation	Description
NLM	Neural Language Model
MLM	Masked Language Modelling
MarianNMT	Marian neural machine translation
MarianMT	Marian machine translation
MarianCG	Marian code generation
API	Application Programming Interface
BLEU	Bilingual evaluation understudy
OPUS	Open Parallel Corpus
NL	Natural language
CoNaLa	Code/natural language dataset
AST	Abstract syntax tree
BPE	Byte-pair-encoding
IRs	Intermediate Representations
LCS	Longest Common Subsequence
LLM	Large Language Model
RLHF	Reinforcement Learning from Human Feedback
SFT	Supervised Fine-Tuning
PEFT	Parameter Efficient Fine Tuning
LoRA	Low-Rank Adaptation of Large Language Models
QLoRA	Quantized Low-Rank Adaptation of Large Language Models

List of Publications

1. Ahmed Soliman, Samir Shaheen, and Mayada Hadhoud (2024). "Leveraging Pre-trained Language Models For Code Generation". In: Complex & Intelligent Systems. <https://doi.org/10.1007/s40747-024-01373-8> [1].
2. Ahmed S. Soliman, Mayada M. Hadhoud, and Samir I. Shaheen (2022) MarianCG: a code generation transformer model inspired by machine translation. Journal of Engineering and Applied Science (JEAS), Volume 69, Article Number 104. <https://doi.org/10.1186/s44147-022-00159-4> [2].
3. Ahmed Soliman, Samir Shaheen, and Mayada Hadhoud (2024). "Effect of Language Models on Programming Assistance Capabilities". Manuscript submitted for publication[3].

Abstract

Quá trình tạo mã là một thách thức then chốt trong phát triển phần mềm hiện đại, liên quan đến việc tạo và duy trì mã tự động dựa trên mô tả và thông số kỹ thuật của con người. Quy trình này về cơ bản thể hiện một dạng diễn giải ngữ nghĩa. Động lực đằng sau việc khám phá quá trình tạo mã nằm ở tiềm năng hỗ trợ các kỹ sư phần mềm và nhà phát triển hiện thực hóa các triển khai thực tế đồng thời giảm thiểu đường cong học tập liên quan đến ngôn ngữ lập trình và công nghệ mới nổi. Các phương pháp phát triển phần mềm truyền thống thường đòi hỏi nhiều công sức để tạo và gỡ lỗi mã, dẫn đến các quy trình kéo dài và dễ xảy ra lỗi.

Trong lĩnh vực mã hóa hỗ trợ AI, tầm quan trọng của việc tạo mã không được đánh giá cao bởi khả năng giải quyết hiệu quả các phức tạp về mã hóa. Điều này có thể thực hiện được thông qua các công cụ tự động phân tích cơ sở mã, hiểu các mẫu lập trình và đưa ra các đề xuất hoặc hoàn thiện mã. Thông qua quá trình đào tạo nghiêm ngặt trên các kho lưu trữ mã mở rộng, các mô hình học máy có được sự hiểu biết sâu sắc về các mẫu, cú pháp và các phương pháp mã hóa tối ưu, phát triển thành các đồng minh có giá trị cho các nhà phát triển. Chúng đưa ra các đề xuất để hoàn thiện mã, xác định các lỗi hoặc lỗ hổng tiềm ẩn và cải tiến mã theo các chuẩn mực mã hóa đã thiết lập. Việc tích hợp tạo mã vào mã hóa hỗ trợ AI hợp lý hóa quy trình phát triển và nâng cao chất lượng mã, cuối cùng dẫn đến cải thiện hiệu quả và độ tin cậy của lập trình.

Luận án này trình bày một cuộc khám phá về việc sử dụng các mô hình ngôn ngữ biến đổi được đào tạo trước trong việc xây dựng các mô hình mã hóa-giải mã để tạo mã. Chúng tôi đã tận dụng việc học chuyển giao để điều chỉnh mô hình dịch máy nhằm giải quyết hiệu quả nhiệm vụ tạo mã, đỉnh cao là tạo ra mô hình MarianCG. Ngoài ra, luận án này đi sâu vào việc hợp nhất nhiều mô hình ngôn ngữ được đào tạo trước, bao gồm DistilRoBERTa, ELECTRA và LUKE, với Marian Decoder để phát triển các mô hình tạo mã.

Hơn nữa, nghiên cứu bao gồm phân tích tỉ mỉ về mã được tạo ra, bao gồm phát hiện lỗi tĩnh và tái cấu trúc sau đó. Việc áp dụng các mô hình ngôn ngữ được đào tạo trước nỗi lên như một chiến lược thông minh để vượt qua các thách thức về tạo mã, dẫn đến việc tạo ra một mô hình dịch máy thành thạo có khả năng chuyển đổi mô tả của con người thành mã thực thi. Các mô hình mới, được hỗ trợ bởi các mô hình ngôn ngữ được đào tạo trước, khéo léo giải quyết các phức tạp của việc tạo mã, tăng cường độ chính xác trong các tác vụ tạo chuỗi.

Cuối cùng, luận án này khám phá việc triển khai QLoRA và PEFT kết hợp với mô hình ngôn ngữ lớn Llama 2 (LLM), xem xét toàn diện nền tảng lý thuyết và ý nghĩa thực tế của chúng đối với việc tạo mã nhiều dòng. Điều này giới thiệu mô hình Llama2-CodeGen, trong đó cuộc điều tra bao gồm phân tích chuyên sâu về tác động của chúng đối với khả năng thích ứng của mô hình, hiệu quả tài nguyên và hiệu suất so với các mô hình tạo mã một dòng thông thường.

Các phát hiện thực nghiệm chứng minh lời hứa của các mô hình ngôn ngữ chuyển đổi được đào tạo trước để tạo mã, mang lại độ chính xác và hiệu quả vượt trội. Các mô hình được đề xuất, được đánh giá trên các tập dữ liệu CoNaLa và DJANGO, xác nhận tính hiệu quả của chúng. Đáng chú ý, mô hình MarianCG thể hiện số liệu BLEU là 34,43 và độ chính xác khớp chính xác là 10,2% trên CoNaLa, đồng thời mang lại kết quả hấp dẫn trên DJANGO với điểm BLEU là 90,41 và độ chính xác khớp chính xác là 81,83%.

Mô hình RoBERTaMarian, sự kết hợp giữa DistilRoBERTa và Marian Decoder, đạt điểm BLEU cao nhất là 35,74 và độ chính xác khớp chính xác là 13,8% trên CoNaLa, nhấn mạnh độ chính xác của nó trong việc tạo mã. LUKEMarian, sự kết hợp giữa LUKE và Marian Decoder, tạo ra kết quả đầy hứa hẹn trên tập dữ liệu DJANGO, đạt điểm BLEU là 89,34 và độ chính xác khớp

chính xác là 78,50%.

Về bản chất, luận án này làm sáng tỏ khả năng của các mô hình ngôn ngữ biến đổi được đào tạo trước để thúc đẩy lĩnh vực tạo mã. Luận án giới thiệu các mô hình sáng tạo có khả năng điều hướng các tình huống mã hóa phức tạp, thể hiện độ chính xác và hiệu quả được nâng cao.

Chapter 1

Introduction

Quá trình cho phép máy nhận dạng một biểu diễn ý nghĩa của nó được gọi là phân tích ngữ nghĩa[4]. Quá trình này chuyển đổi các mô tả của con người về lời nói thành các biểu diễn dạng logic. Dịch máy, trả lời câu hỏi, quy nạp bản thể, lý luận tự động và tạo mã đều là các ví dụ về ứng dụng phân tích ngữ nghĩa [5] [6] [7].

Phân tích ngữ nghĩa và tạo mã là các khái niệm gắn bó chặt chẽ trong lĩnh vực Xử lý dựa trên bộ chuyển đổi. Phân tích ngữ nghĩa liên quan đến việc chuyển đổi các biểu thức của con người thành các biểu diễn có cấu trúc nắm bắt được ý nghĩa hoặc mục đích của đầu vào [8]. Mặt khác, tạo mã liên quan đến quá trình tự động tạo mã thực thi dựa trên một đặc điểm kỹ thuật hoặc mô tả nhất định[9].

Phân tích ngữ nghĩa đóng vai trò là một bước cơ bản trong quá trình tạo mã, vì nó thu hẹp khoảng cách giữa hiểu ngôn ngữ và tổng hợp mã [10]. Nó liên quan đến việc ánh xạ ngữ nghĩa của đầu vào ngôn ngữ tự nhiên thành một biểu diễn chính thức có thể được xử lý và chuyển đổi thêm thành mã thực thi[11]. Biểu diễn này có thể ở dạng Cây cú pháp trừu tượng (AST), Biểu diễn trung gian (IR) hoặc các cấu trúc phù hợp khác để nắm bắt chức năng mong muốn của chương trình như thể hiện trong Hình 1.1.

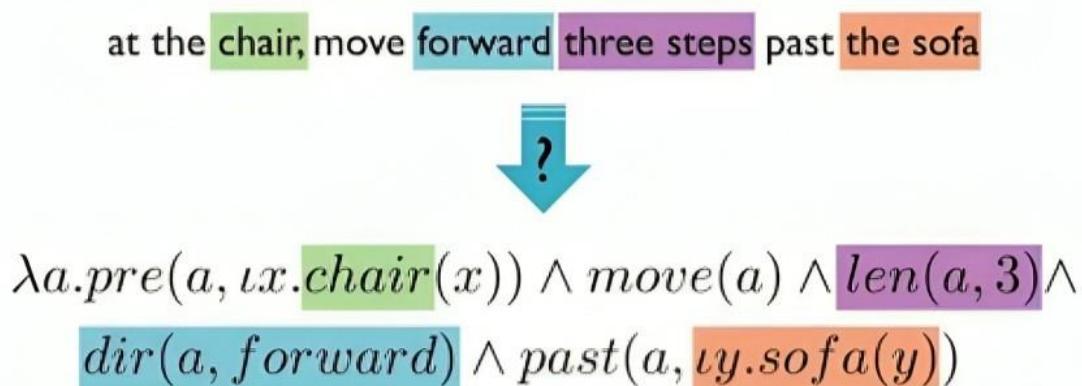


Figure 1.1: Example of Semantic Parsing

Đổi lại, việc tạo mã sử dụng các biểu diễn có cấu trúc bắt nguồn từ phân tích ngữ nghĩa để tự động tạo và duy trì mã [12] [13] [14]. Bằng cách tận dụng nhiều kỹ thuật khác nhau, chẳng hạn như thuật toán học máy, phương pháp tiếp cận dựa trên mẫu hoặc chuyển đổi dựa trên quy tắc, các hệ thống tạo mã hướng đến mục tiêu tạo ra mã phản ánh chính xác chức năng mong muốn được mô tả bằng ngôn ngữ tự nhiên

[15] [16] [17] [18]. Những thách thức trong việc tạo mã không chỉ nằm ở việc nắm bắt chính xác ngữ nghĩa của đầu vào mà còn ở việc đảm bảo mã được tạo ra có cú pháp chính xác, hiệu quả và tuân thủ các tiêu chuẩn mã hóa [19].

Các nhà nghiên cứu và học viên khám phá các phương pháp khác nhau, bao gồm việc sử dụng các mô hình ngôn ngữ được đào tạo trước, học chuyển giao và các kỹ thuật học sâu, để nâng cao độ chính xác và hiệu quả của các hệ thống tạo mã. Do đó, phân tích ngữ nghĩa và tạo mã có liên kết chặt chẽ với nhau, trong đó phân tích ngữ nghĩa tạo thành nền tảng để tạo mã thể hiện trung thực chức năng dự định được thể hiện trong các mô tả ngôn ngữ tự nhiên [20].

Tạo mã là một lĩnh vực quan trọng có thể dự đoán và tạo mã từ các đầu vào ngôn ngữ tự nhiên [21] như thể hiện trong Hình.1.2. Bằng cách kết hợp các công cụ tạo mã cung cấp độ chính xác và tối ưu hóa cao hơn, năng suất của các công cụ lập trình có thể được nâng cao [22]. Các nhà nghiên cứu đã có những đóng góp đáng kể vào việc phát triển các mô hình tự động tạo mã từ các mô tả ngôn ngữ tự nhiên, cho phép học và sản xuất mã [23].

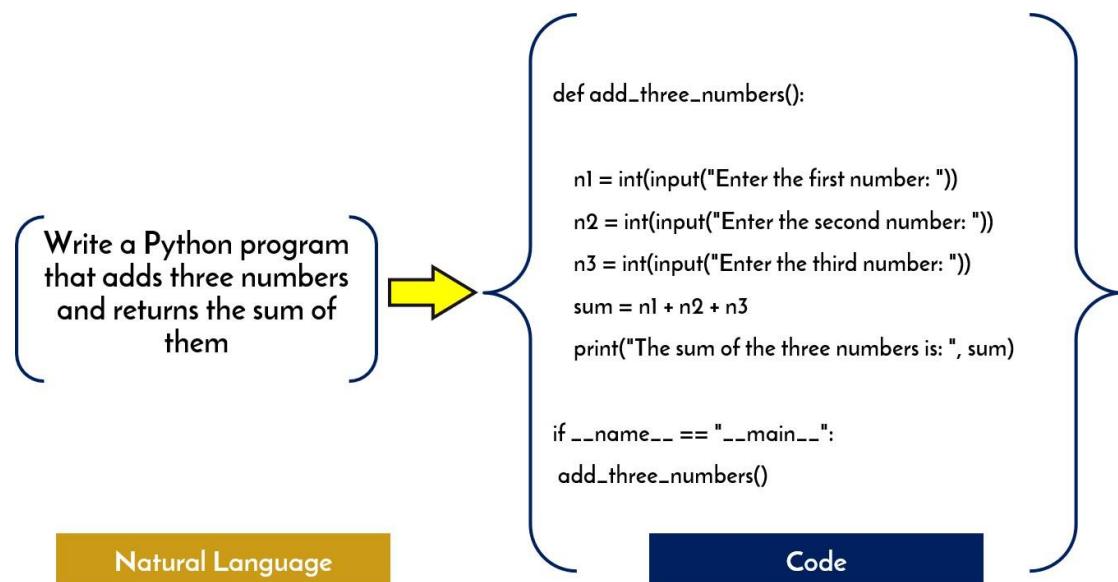


Figure 1.2: Example of Code Generation

Tạo mã đã trở thành một khía cạnh then chốt trong lĩnh vực lập trình, nhằm mục đích hợp lý hóa và tạo điều kiện thuận lợi cho quá trình mã hóa cho cả nhà phát triển dày dạn kinh nghiệm và người mới bắt đầu. Nó liên quan đến việc tạo tự động các đoạn mã hoặc chuỗi mã hoàn chỉnh dựa trên mô tả ngôn ngữ tự nhiên hoặc thông số kỹ thuật

cấp cao. Nhu cầu về các công cụ tạo mã hiệu quả đã tăng lên đáng kể, do tính phức tạp ngày càng tăng của các ứng dụng phần mềm và nhu cầu phát triển nhanh chóng.

Khái niệm tạo mã xoay quanh việc chuyển đổi các mô tả dễ đọc của con người thành mã thực thi, thu hẹp khoảng cách giữa hiểu ngôn ngữ tự nhiên và ngôn ngữ lập trình. Khi độ phức tạp và quy mô của các dự án phần mềm tiếp tục mở rộng, việc mã hóa thủ công trở nên tốn nhiều công sức và dễ xảy ra lỗi. Tạo mã tìm cách giải quyết những thách thức này bằng cách tận dụng sức mạnh của máy học và xử lý dựa trên bộ chuyển đổi để tự động tạo ra các giải pháp mã chính xác và đáng tin cậy.

Một trong những lợi ích chính của việc tạo mã nằm ở khả năng nâng cao năng suất của nhà phát triển và giảm thời gian phát triển [24]. Bằng cách tự động hóa các tác vụ mã hóa lặp đi lặp lại và cung cấp các đoạn mã được điều chỉnh theo các yêu cầu cụ thể, tạo mã cho phép các lập trình viên tập trung vào các khía cạnh thiết kế và giải quyết vấn đề cấp cao hơn [25]. Ngoài ra, tạo mã đóng vai trò quan trọng trong việc thúc đẩy tính nhất quán của mã và tuân thủ các tiêu chuẩn mã hóa, tạo ra các hệ thống phần mềm dễ bảo trì và mạnh mẽ hơn.

Khi các cơ sở mã và thư viện ngày càng lớn hơn và đa dạng hơn, việc tạo mã ngày càng có giá trị trong việc đơn giản hóa việc tái sử dụng và tích hợp mã. Nó tạo điều kiện thuận lợi cho việc kết hợp các mô-đun mã và thư viện hiện có vào các dự án mới, giúp giảm hiệu quả sự trùng lặp và thúc đẩy tính mô-đun của mã [26]. Hơn nữa, việc tạo mã góp phần vào quá trình dân chủ hóa phát triển phần mềm, cho phép những cá nhân có trình độ chuyên môn về mã hóa khác nhau tham gia vào quá trình tạo ra các giải pháp phần mềm.

Luận văn này giới thiệu các mô hình tạo mã và đi sâu vào ý nghĩa và sự phức tạp của việc tạo mã, khám phá nhiều kỹ thuật và mô hình khác nhau để nâng cao khả năng của nhiệm vụ này. Bằng cách sử dụng các mô hình ngôn ngữ chuyển đổi được đào tạo trước và sử dụng các kỹ thuật tinh chỉnh, vấn đề tạo mã có thể được giải quyết hiệu quả [27]. Thông qua một loạt các thử nghiệm và đánh giá, chúng tôi hướng đến mục tiêu phát triển các mô hình tạo mã tiên tiến đáp ứng các thách thức về mã hóa trong thế giới thực, trao quyền cho các nhà phát triển các giải pháp tạo mã hiệu quả và hiệu suất.

1.1 Motivation

Viết và triển khai mã là một phần thiết yếu của vòng đời phát triển phần mềm [28]. Đây là quy trình chuyển đổi phân tích và thiết kế trong phát triển phần mềm thành các phương pháp kỹ thuật để thực hiện nhiệm vụ mong muốn. Nhiệm vụ tạo mã có một số thách thức nhất định vì đầu ra có cấu trúc được xác định rõ ràng và miễn, cấu trúc của đầu vào và cấu trúc của đầu ra không giống nhau. Vì viết mã là hợp lý và có cấu trúc, khả năng viết chương trình và mã đã trở thành một kỹ năng có giá trị được coi trọng như việc học một ngôn ngữ khác.

Trợ lý mã đã đạt được tiến bộ đáng kể trong những năm gần đây, được hưởng lợi từ những tiến bộ trong thuật toán học máy, xử lý dựa trên bộ biến đổi và các kỹ

thuật phân tích phần mềm. Tiến bộ này đã tạo ra các mô hình AI và công cụ phân tích mã có khả năng ẩn tượng trong việc hiểu ngữ nghĩa mã, tạo ra các đề xuất mã chính xác và dự đoán các mẫu mã dựa trên các tín hiệu theo ngữ cảnh. Những tiến bộ này có tiềm năng cách mạng hóa quy trình phát triển phần mềm, dẫn đến tăng năng suất, nâng cao chất lượng mã và cải thiện sự hợp tác trong các nhóm phát triển.

Tuy nhiên, vẫn còn nhiều thách thức trong lĩnh vực tạo mã. Những thách thức này bao gồm nhu cầu về dữ liệu đào tạo đáng tin cậy, giải quyết các thành kiến có trong kho lưu trữ mã và tạo ra sự cân bằng tinh tế giữa tự động hóa và sự sáng tạo của con người trong quá trình phát triển. Hơn nữa, hiệu quả của các mô hình ngôn ngữ được đào tạo trước trong việc xây dựng các mô hình tạo mã và các lợi ích tiềm năng của việc kết hợp nhiều mô hình khác nhau để nâng cao hiệu suất đòi hỏi phải khám phá và đánh giá thêm.

Tóm lại, luận án được thúc đẩy bởi việc khám phá và sử dụng các mô hình ngôn ngữ được đào tạo trước để thúc đẩy các hoạt động tạo mã. Luận án tìm cách giải quyết các thách thức trong mã hóa hỗ trợ AI và mở rộng ranh giới của các kỹ thuật tạo mã, dẫn đến năng suất và chất lượng mã được cải thiện.

1.2 Proposed Solution

Luận án này trình bày một giải pháp toàn diện nhằm nâng cao quy trình phát triển phần mềm thông qua phương pháp tiếp cận đa diện. Giải pháp được đề xuất kết hợp sức mạnh của hiểu ngôn ngữ tự nhiên, tạo mã tự động, các kỹ thuật kiểm tra lỗi tiên tiến và các công cụ tái cấu trúc để hợp lý hóa vòng đời phát triển phần mềm. Luận án được thúc đẩy bởi mục tiêu giải quyết những thách thức mà các nhà phát triển phải đối mặt và đóng góp vào sự tiến bộ của việc tạo mã AI. Luận án tìm cách điều tra ý nghĩa của các mô hình ngôn ngữ chuyển đổi được đào tạo trước trong việc xây dựng các mô hình tạo mã.

Mục tiêu là khắc phục các trở ngại liên quan đến việc tạo mã và phát triển một mô hình dịch máy có khả năng chấp nhận đầu vào ngôn ngữ tự nhiên và tạo ra đầu ra mã tương ứng. Ngoài ra, luận án có ý định tinh chỉnh mã đã tạo bằng cách kết hợp các công cụ sửa lỗi và các kỹ thuật cải tiến mã. Quy trình này đảm bảo rằng mã đã tạo đáp ứng các tiêu chuẩn mã hóa Python bắt buộc và các thông lệ tốt nhất. Bằng cách đó, luận án góp phần vào độ tin cậy, khả năng bảo trì và khả năng đọc tổng thể của cơ sở mã, cuối cùng tạo điều kiện thuận lợi cho việc phát triển phần mềm hiệu quả.

Quy trình và đường ống được đề xuất cho hệ thống tạo mã AI được đề xuất được hiển thị trong Hình 1.3 và bao gồm ba giai đoạn và đường ống riêng biệt tạo thành một phương pháp tiếp cận toàn diện để tạo mã và hệ thống tạo mã AI, bao gồm đầu vào ngôn ngữ tự nhiên, phân tích mã và sửa lỗi..

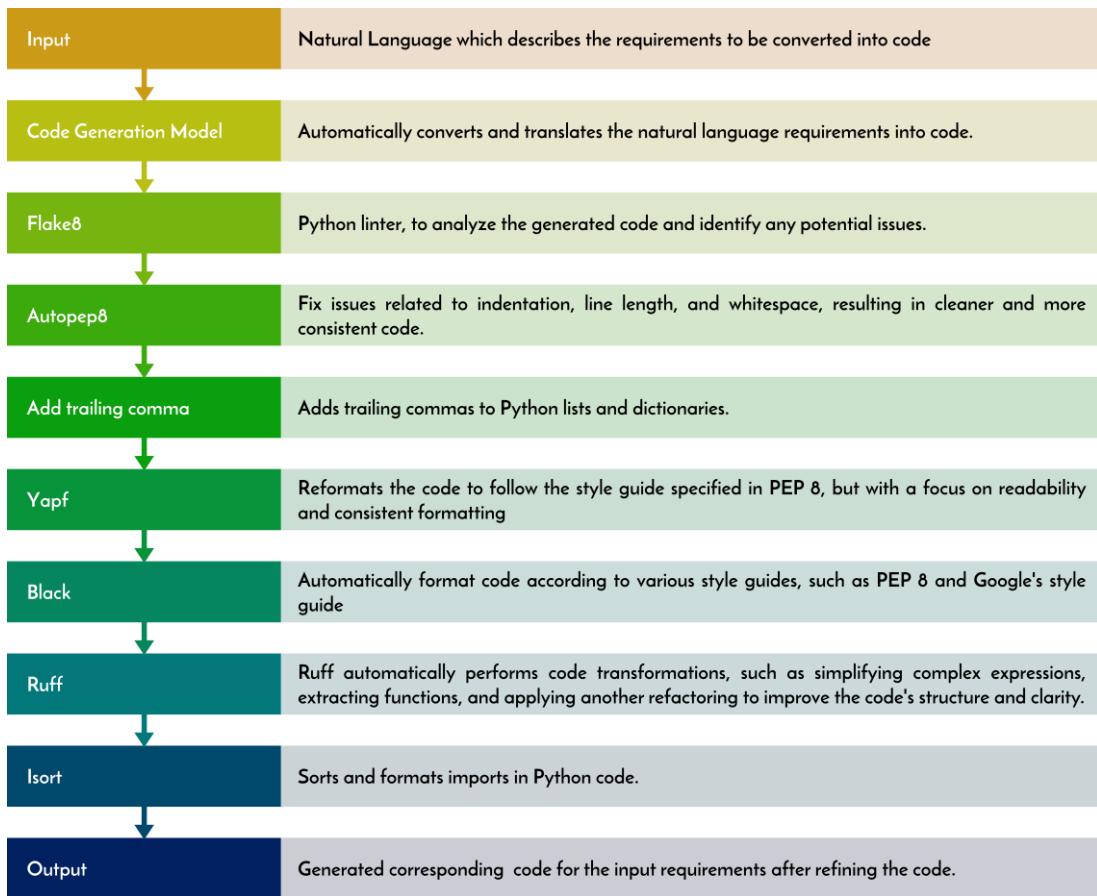


Figure 1.3: Pipelines for the Code Assistance System

Sau đây là các thành phần chính của giải pháp được đề xuất cùng với vai trò của chúng trong việc cách mạng hóa cách phần mềm được khai niêm hóa và phát triển:

- Tạo mã từ ngôn ngữ tự nhiên

Phát triển phần mềm hiện đại đòi hỏi sự giao tiếp và hợp tác rõ ràng giữa các nhà phát triển và các bên liên quan. Tuy nhiên, khoảng cách giữa ngôn ngữ kỹ thuật của mã và ngôn ngữ tự nhiên được các bên liên quan không phải kỹ thuật sử dụng có thể dẫn đến hiểu lầm và kém hiệu quả. Để thu hẹp khoảng cách này, đường ống đầu tiên của giải pháp được đề xuất tập trung vào việc tạo mã trực tiếp từ các thông số kỹ thuật ngôn ngữ tự nhiên.

Quá trình này bao gồm việc đào tạo một mô hình ngôn ngữ để hiểu các mô tả ngôn ngữ tự nhiên về các yêu cầu phần mềm và dịch chúng thành các đoạn mã có thể thực thi. Bằng cách tự động hóa bản dịch này, giải pháp này nhằm mục đích nâng cao độ chính xác và tốc độ tạo mã ban đầu, giúp phát triển phần mềm dễ tiếp cận hơn với nhiều cá nhân hơn, bao gồm cả những người không có nền tảng lập trình sâu rộng.

- Linting và Phân tích lỗi bằng Flake8

Khi các dự án phần mềm phát triển về quy mô và độ phức tạp, khả năng đưa vào lỗi và sự không nhất quán cũng tăng lên. Các công cụ linting truyền thống đã chứng minh được

hiệu quả trong việc xác định các vấn đề về cú pháp và phong cách trong mã.

Tuy nhiên, các công cụ này thường không có khả năng phát hiện các lỗi logic tinh vi hơn và các lỗi thiết kế có thể dẫn đến các vấn đề về thời gian chạy.

Đường ống thứ hai của giải pháp được đề xuất tận dụng Flake8, một công cụ kiểm tra lỗi được sử dụng rộng rãi, để thực hiện phân tích toàn diện về mã được tạo. Không chỉ làm nổi bật các lỗi cú pháp, đường ống này còn đi sâu vào việc xác định các lỗi logic tiềm ẩn, các mẫu mã không phù hợp và tuân thủ các thông lệ tốt nhất. Bằng cách tích hợp phân tích tỉ mỉ này vào quy trình phát triển, các nhà phát triển có thể chủ động giải quyết các vấn đề trước khi chúng lan truyền và tác động đến chức năng của phần mềm.

- Tái cấu trúc và sửa lỗi bằng các công cụ nâng cao

Ngay cả khi tạo mã hiệu quả và kiểm tra lỗi tỉ mỉ, các dự án phần mềm vẫn gặp phải những thách thức đòi hỏi phải cải tiến và nâng cao theo từng bước. Đường ống thứ ba của giải pháp được đề xuất giới thiệu một bộ công cụ nâng cao để tái cấu trúc và sửa lỗi. Các công cụ này được thiết kế để đề xuất các sửa đổi thông minh cho cấu trúc mã, cải thiện khả năng đọc mã và khắc phục các lỗi thiết kế.

Bằng cách tự động hóa quy trình tái cấu trúc và sửa lỗi, đường ống này đẩy nhanh chu kỳ bảo trì phần mềm. Các nhà phát triển có thể tin thực hiện các thay đổi, biết rằng các công cụ đã phân tích kỹ lưỡng tác động và đưa ra các đề xuất khả thi. Quá trình lặp đi lặp lại này không chỉ tăng cường tính mạnh mẽ của phần mềm mà còn góp phần vào các hoạt động phát triển hiệu quả và nhanh nhẹn hơn.

Giải pháp được đề xuất, bao gồm việc tạo mã từ ngôn ngữ tự nhiên, kiểm tra lỗi và phân tích lỗi chuyên sâu với Flake8, cùng các công cụ sửa lỗi và tái cấu trúc thông minh, đưa ra một cách tiếp cận toàn diện để cách mạng hóa quy trình phát triển phần mềm. Bằng cách giải quyết trực tiếp các thách thức về giao tiếp, chất lượng mã và bảo trì, giải pháp này nhằm mục đích trao quyền cho các nhà phát triển để tạo ra các hệ thống phần mềm đáng tin cậy, hiệu quả và dễ bảo trì hơn. Các chương tiếp theo sẽ đi sâu vào các chi tiết kỹ thuật của từng đường ống, chứng minh các điểm mạnh riêng lẻ và tác động hiệp đồng của chúng đối với vòng đời phát triển phần mềm.

1.3 Thesis Contribution

Luận án này đóng góp đáng kể vào lĩnh vực tạo mã thông qua những phát hiện chính sau:

1. Chứng minh tính hiệu quả của Mô hình dịch máy để hoạt động như một mô hình tạo mã. Trong khuôn khổ này, chúng tôi giới thiệu mô hình biến đổi MarianCG, được thiết kế riêng cho việc tạo mã từ ngôn ngữ tự nhiên.
2. Đề xuất các mô hình lai mới để tạo mã bằng cách tận dụng các mô hình ngôn ngữ được đào tạo trước như BERT, RoBERTa và Marian. Chúng tôi đề xuất các mô hình lai mới để giải quyết vấn đề tạo mã. Các mô hình được đề xuất đạt được độ chính xác cao trong tác vụ tạo mã trên các tập dữ liệu CoNaLa và DJANGO.
3. Làm nổi bật tầm quan trọng của các mô hình ngôn ngữ được đào tạo trước.

Nghiên cứu của chúng tôi nhấn mạnh đến sự cần thiết của các bộ mã hóa được đào tạo trước trong các tác vụ hình thành chuỗi. Hơn nữa, chúng tôi chứng minh rằng việc chia sẻ trọng số giữa bộ mã hóa và bộ giải mã thường có lợi.

4. Tiến hành phân tích lỗi và tinh chỉnh mã đã tạo. Chúng tôi tiến hành phân tích lỗi toàn diện của mã đã tạo và tinh chỉnh để tuân thủ các tiêu chuẩn mã Python. Quy trình này đảm bảo mã đã tạo đáp ứng các yêu cầu về chất lượng và tính nhất quán.

5. Giới thiệu mô hình Llama2-CodeGen Mô hình Llama2-CodeGen để tạo mã. Điều này được thực hiện bằng cách tích hợp các phương pháp QLoRA và PEFT với mô hình ngôn ngữ lớn Llama 2 (LLM) và kỹ thuật nhanh chóng với các kỹ thuật AI tạo sinh, dẫn đến sự phát triển của mô hình Llama2-CodeGen.

Thông qua những đóng góp này, nghiên cứu của chúng tôi góp phần thúc đẩy lĩnh vực tạo mã, chứng minh hiệu quả của các mô hình dịch máy, tầm quan trọng của bộ mã hóa được đào tạo trước và ý nghĩa của phân tích lỗi và tinh chỉnh trong việc đạt được khả năng tạo mã đáng tin cậy và chuẩn hóa.

1.4 Thesis outline

- Luận văn này được tổ chức như sau:
 - • Chương 2 đánh giá các kỹ thuật và phương pháp khác nhau trong học sâu và đào tạo trước các mô hình lớn.
 - • Chương 3 trình bày các công trình liên quan và cách các nhà nghiên cứu khác giải quyết vấn đề tạo mã.
 - • Chương 4 được chia thành hai phần, phần đầu tiên, trình bày mô hình tạo mã MarianCG. Phần thứ hai, tận dụng các mô hình ngôn ngữ được đào tạo trước với Marian Decoder để tạo các mô hình tạo mã mã hóa-giải mã.
 - • Chương 5 cung cấp các kết quả thử nghiệm.
 - • Chương 6 trình bày trường hợp sử dụng cho tác vụ tạo mã nhiều dòng bằng cách tinh chỉnh mô hình Llama-2 như một mô hình ngôn ngữ lớn với các kỹ thuật PEFT và QLoRA và AI tạo sinh.
 - • Cuối cùng, kết luận và đề xuất công trình trong tương lai được trình bày.

Chapter 2

Background

Chương này đi sâu vào giao điểm của học sâu và xử lý dựa trên bộ biến đổi, khám phá cách biểu diễn các từ để xử lý trong các mô hình học sâu. Ngoài ra, chương này đi sâu vào các mô hình trình tự và cung cấp thông tin chi tiết về việc đánh giá các mô hình seq2seq. Hơn nữa, chương này cung cấp phần giới thiệu về các mô hình bộ biến đổi và xem xét cách học chuyển giao có thể cải thiện hiệu suất trong các tác vụ đầy thách thức thông qua việc sử dụng các mô hình được đào tạo trước. Cuối cùng, chương này giải quyết việc phân tích lỗi trong mã Python và cung cấp hướng dẫn về việc sửa các vấn đề cú pháp trong mã được tạo ra để đảm bảo tuân thủ các tiêu chuẩn mã hóa.

2.1 Deep Learning and Transformer-Based Processing

Những tiến bộ gần đây trong học sâu [29] đã cho phép các nhà nghiên cứu phát triển các mô hình có thể học từ lượng dữ liệu khổng lồ và tạo ra đầu ra có ý nghĩa, biểu cảm. Các kiến trúc và thuật toán học sâu đã có những tiến bộ ấn tượng trong các lĩnh vực như thị giác máy tính và nhận dạng mẫu. Theo xu hướng này, nghiên cứu NLP gần đây hiện đang ngày càng tập trung vào việc sử dụng các phương pháp học sâu mới [30] [31] [32] [33]. Xử lý dựa trên máy biến áp đã trở thành một lĩnh vực ngày càng quan trọng và các kỹ thuật học sâu đã đóng một vai trò quan trọng trong việc thúc đẩy các tác vụ tiên tiến.

Tạo mã là một trong những ứng dụng được hưởng lợi từ học sâu, vì nó có thể cải thiện năng suất, tính nhất quán và chất lượng mã [34]. Mặc dù các công cụ tạo mã có thể hữu ích, nhưng chúng nên được sử dụng kết hợp với sự giám sát và đánh giá của con người để đảm bảo rằng mã được tạo ra đáp ứng các tiêu chuẩn cần thiết về chất lượng và bảo mật. Ngoài việc tạo mã, các mô hình ngôn ngữ AI cũng có thể được sử dụng để cải thiện chất lượng của các bài báo khoa học bằng cách cung cấp các kiểm tra ngôn ngữ và ngữ pháp và hỗ trợ soạn thảo các đoạn văn [35]. Khi học sâu tiếp tục phát triển, có khả năng chúng ta có thể thấy nhiều ứng dụng NLP sáng tạo hơn với học sâu. Học sâu cũng đã cho thấy kết quả đầy hứa hẹn trong nhiều ứng dụng hơn, chẳng hạn như dịch máy và nhận dạng giọng nói [36]. Vẫn còn nhiều thách thức cần vượt qua, chẳng hạn như xử lý sự mờ hổ và ngữ cảnh.

2.1.1 Deep Learning

Học sâu là một tập hợp con của học máy lấy cảm hứng từ cấu trúc sinh học của não người và cách thức hoạt động của nó. Về cơ bản, nó là một mạng nơ-ron có ba lớp trở lên[37]. Ngày nay, mạng nơ-ron là một trong những thuật toán học máy nổi bật nhất vượt trội hơn các kỹ thuật chính xác và tốc độ khác [38]. Khả năng học sâu đề cập đến khả năng của các mô hình AI trong việc học và đưa ra dự đoán về các mẫu phức tạp và trừu tượng trong dữ liệu thông qua việc sử dụng các thuật toán và kiến trúc mạng nơ-ron tiên tiến [39]. Kiến trúc của mạng nơ-ron sâu được thể hiện trong Hình 2.1.

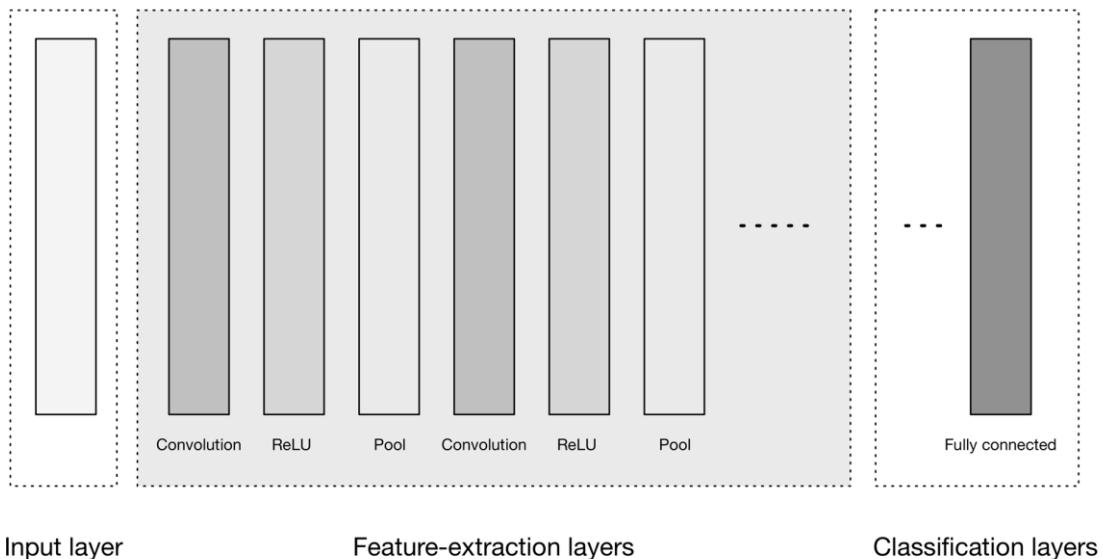


Figure 2.1: Deep Neural Network Architecture [40]

Trong khi mạng nơ-ron một lớp vẫn có thể tạo ra các dự đoán gần đúng, nhiều lớp ẩn hơn có thể tối ưu hóa và điều chỉnh đáng kể để có độ chính xác. Điều thú vị nhất là một mô hình học sâu duy nhất có thể học nghĩa của từ và thực hiện các tác vụ ngôn ngữ, tránh nhu cầu thực hiện các tác vụ ngôn ngữ phức tạp [40]. Nhiều ứng dụng và dịch vụ trí tuệ nhân tạo (AI) dựa vào học sâu để tăng cường tự động hóa bằng cách thực hiện các hoạt động phân tích và vật lý mà không cần tương tác của con người [41]. Công nghệ học sâu là cốt lõi của cả các sản phẩm và dịch vụ được biết đến rộng rãi và các cải tiến sắp tới [42].

2.1.2 Transformer-Based Processing

Học sâu và Xử lý dựa trên máy biến áp [43] đã nổi lên như hai lĩnh vực đột phá đi đầu trong nghiên cứu trí tuệ nhân tạo [44]. Học sâu, một tập hợp con của học máy, tập trung vào việc đào tạo các mạng nơ-ron nhân tạo với nhiều lớp để học các biểu diễn phân cấp của dữ liệu [42]. Ngoài ra, nhiều nhiệm vụ liên quan đến việc phân tích, hiểu và tạo ra ngôn ngữ của con người bằng các hệ thống tính toán.

Việc tích hợp các kỹ thuật học sâu với xử lý văn bản [45] đã cách mạng hóa cách chúng ta xử lý, diễn giải và tạo ra ngôn ngữ tự nhiên.

Lượng lớn dữ liệu văn bản phi cấu trúc có sẵn trên internet, các nền tảng truyền thông xã hội và các nguồn khác đã đòi hỏi các phương pháp tiên tiến để trích xuất những hiểu biết có ý nghĩa từ thông tin này [46]. Các mô hình học sâu đã cho thấy khả năng đặc biệt trong việc xử lý sự phức tạp vốn có của ngôn ngữ, cho phép máy móc hiểu và tạo ra văn bản giống con người [41]. Với khả năng học hỏi từ dữ liệu quy mô lớn, các mô hình học sâu đã vượt qua các phương pháp tiếp cận dựa trên quy tắc truyền thống trong nhiều tác vụ NLP khác nhau, bao gồm phân tích tình cảm, dịch máy, tạo văn bản và trả lời câu hỏi.

Xử lý dựa trên Transformer là một loại xử lý và phân tích dữ liệu dựa trên một kiến trúc học sâu cụ thể được gọi là Transformer. Kiến trúc Transformer được giới thiệu trong một bài báo có tính đột phá có tựa đề "Attention Is All You Need"

[30] của Vaswani và cộng sự vào năm 2017 và kể từ đó đã trở thành một khối xây dựng cơ bản cho nhiều tác vụ xử lý ngôn ngữ tự nhiên (NLP) và học máy khác nhau[43].

Về bản chất, kiến trúc Transformer sử dụng một cơ chế được gọi là tự chú ý để xử lý các chuỗi dữ liệu, chẳng hạn như câu hoặc dữ liệu chuỗi thời gian, song song thay vì tuần tự. Sự song song hóa này cho phép xử lý các chuỗi hiệu quả và hiệu suất hơn, khiến nó phù hợp với nhiều tác vụ ngoài NLP.

2.1.3 Deep Learning Techniques for Language Representation

Các thuật toán Học máy và Học sâu chỉ hoạt động trên dữ liệu đầu vào số. Vì vậy, việc biểu diễn các từ và câu theo cách nắm bắt được ý nghĩa ngữ nghĩa và ngữ cảnh của chúng được coi là một trong những thách thức cơ bản trong xử lý văn bản [47] [48] [49]. Để biểu diễn chuỗi câu đầu vào trong các mô hình này, có một số kỹ thuật có sẵn để mã hóa các từ. Các kỹ thuật này bao gồm một loạt các phương pháp:

- Mã hóa One-Hot

Mỗi từ được biểu diễn dưới dạng vectơ nhị phân trong đó chỉ có một phần tử được đặt thành 1, biểu thị sự hiện diện của từ đó. Sơ đồ mã hóa này tạo ra biểu diễn thừa thót và có chiều cao [50].

- Nhúng từ:

Các mô hình học sâu đã giới thiệu các kỹ thuật mới như nhúng từ, biến đổi các từ thành các biểu diễn số dày đặc nắm bắt các mối quan hệ ngữ nghĩa [51]. Nhúng từ được đào tạo trên các tập hợp văn bản lớn và có thể nắm bắt các thuộc tính cú pháp và ngữ nghĩa của từ [52]. Chúng đã được sử dụng trong nhiều tác vụ NLP, bao gồm phân tích tình cảm, dịch máy và truy xuất thông tin.

Các kỹ thuật phổ biến như Word2Vec [53], GloVe [54] và FastText [55] học các biểu diễn vectơ liên tục bằng cách xem xét ngữ cảnh và các mẫu đồng xuất hiện của các từ.

- Nhúng ngữ cảnh:

Nhúng ngữ cảnh là sự phát triển của nhúng từ. Trái ngược với nhúng từ, nắm bắt ý nghĩa của một từ một cách riêng lẻ, nhúng ngữ cảnh nắm bắt ý nghĩa của các từ trong ngữ cảnh [56]. Điều này đạt được bằng cách đào tạo nhúng trên các câu thay vì các từ riêng lẻ [57]. Nhúng từ ngữ cảnh, chẳng hạn như nhúng được tạo bởi các mô hình như ELMo [58], GPT [59] và BERT [60], tính đến ngữ cảnh xung quanh để tạo ra các biểu diễn từ [61].

Các nhúng này không chỉ nắm bắt ý nghĩa ngữ nghĩa của từ mà còn nắm bắt thông tin ngữ cảnh của từ trong một câu hoặc tài liệu nhất định [62]. Nhúng ngữ cảnh có thể nắm bắt các đặc điểm của ngôn ngữ như sự mơ hồ và mỉa mai. Chúng đã được chứng minh là có hiệu quả cao trong việc hiểu ngôn ngữ tự nhiên và đã được áp dụng cho các tác vụ nhu dịch máy, phân tích tình cảm và phân loại văn bản [63].

- Biểu diễn từ phụ

Thay vì biểu diễn toàn bộ từ, có thể sử dụng các đơn vị từ phụ như n-gram ký tự hoặc mã hóa cặp byte (BPE) [64]. Các biểu diễn này cho phép xử lý các từ ngoài vốn từ vựng và nắm bắt hiệu quả thông tin hình thái và cú pháp.

- Các phương pháp tiếp cận kết hợp

Một số mô hình kết hợp nhiều kỹ thuật được đề cập ở trên để nắm bắt cả thông tin cấp độ từ và cấp độ từ phụ [65]. Ví dụ, mô hình Transformer, cơ sở của BERT và GPT, sử dụng nhúng từ cùng với mã hóa vị trí để xử lý chuỗi văn bản [31].

Các biểu diễn này cho phép các mô hình học sâu xử lý và hiểu ngôn ngữ tự nhiên hiệu quả bằng cách nắm bắt thông tin ngữ nghĩa, ngữ cảnh và cấu trúc của từ, do đó cho phép thực hiện các tác vụ nhu phân loại văn bản, phân tích tình cảm, dịch máy, v.v.

Nhìn chung, sự phát triển của các kỹ thuật học sâu như nhúng từ và nhúng ngữ cảnh đã cách mạng hóa lĩnh vực NLP, cho phép biểu diễn ngôn ngữ chính xác và sắc thái hơn. Các kỹ thuật này đã cho phép máy móc hiểu và phân tích ngôn ngữ của con người tốt hơn, mở ra nhiều khả năng cho tương lai của NLP.

Sự ra đời của các mô hình biến đổi trong Xử lý ngôn ngữ tự nhiên (NLP), chẳng hạn như BERT, GPT và các mô hình khác, thực sự đã đưa lĩnh vực này lên một tầm cao mới. Các mô hình ngôn ngữ lớn này đã cho phép các hệ thống NLP thực hiện nhiều tác vụ khác nhau, chẳng hạn như phân loại văn bản, nhận dạng thực thể, phân tích tình cảm và trả lời câu hỏi, với hiệu suất tiên tiến [66].

Các mô hình biến đổi đã được đào tạo trước trên lượng dữ liệu khổng lồ, chẳng

hạn như sách, bài viết và trang web, bằng cách sử dụng các kỹ thuật học không giám sát để tìm hiểu cấu trúc thống kê của ngôn ngữ. Sau đó, chúng có thể được tinh chỉnh trên các tác vụ cụ thể với lượng dữ liệu được gắn nhãn tương đối nhỏ, giúp chúng có thể thích ứng với nhiều ứng dụng trong thế giới thực. Các máy biến áp cũng đã thúc đẩy nghiên cứu về các kiến trúc và mục tiêu đào tạo mới, chẳng hạn như XLNet, RoBERTa, T5 và GShard, nhằm cải thiện hiệu suất, hiệu quả và độ mạnh mẽ của các mô hình ngôn ngữ lớn[67].

Các mô hình máy biến áp tận dụng các cơ chế tự chú ý để nắm bắt các mối quan hệ phụ thuộc giữa các từ và cho phép hiểu ngữ cảnh trên các chuỗi văn bản dài. Điều này cho phép chúng thể hiện tốt hơn ý nghĩa của ngôn ngữ và hoạt động tốt trên nhiều tác vụ NLP. Các mô hình máy biến áp đã chứng minh hiệu suất chưa từng có trong nhiều tác vụ NLP khác nhau và đã trở thành nền tảng của việc hiểu và tạo ngôn ngữ tự nhiên hiện đại [68]. Khung máy biến áp cho phép xử lý song song các chuỗi văn bản, giải quyết hiệu quả các mối quan hệ phụ thuộc từ xa trong mô hình hóa ngôn ngữ.

Khám phá sự hội tụ hấp dẫn của học sâu và NLP mang đến cơ hội hấp dẫn để thúc đẩy tiến bộ trong việc hiểu ngôn ngữ, tạo và tự động hóa. Việc khám phá những hiểu biết có giá trị từ sự tích hợp của học sâu và NLP có tiềm năng to lớn cho những tiến bộ mang tính chuyển đổi trên nhiều lĩnh vực khác nhau [69]. Những điều này bao gồm nhưng không giới hạn ở dịch ngôn ngữ, tạo nội dung, trợ lý ảo và tạo điều kiện cho các tương tác thông minh giữa con người và máy tính.

2.2 Language Modeling

Công việc dự đoán từ hoặc ký tự tiếp theo trong một tài liệu được gọi là mô hình hóa ngôn ngữ. Đây là nhiệm vụ gán xác suất cho các câu trong một ngôn ngữ [70]. Phương pháp này có thể được sử dụng để đào tạo các mô hình ngôn ngữ [71], sau đó có thể được sử dụng cho nhiều tác vụ ngôn ngữ tự nhiên như tạo văn bản, tóm tắt và dịch máy. Ví dụ, nếu chúng ta có một số văn bản chứa nội dung sau:

$$x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(T-2)}, x^{(T-1)}, x^{(T)}$$

khi đó xác suất của văn bản này (theo Mô hình ngôn ngữ) được thể hiện trong Công thức 2.1 và Công thức 2.2.

$$P(x^{(1)}, \dots, x^{(T)}) = P(x^{(1)}) * P(x^{(2)}|x^{(1)}) * \dots * P(x^{(T)}|x^{(T-1)}, \dots, x^{(1)}) \quad (2.1)$$

$$P(x^{(1)}, \dots, x^{(T)}) = \prod_{t=1}^T P(x^{(t)}|x^{(t-1)}, \dots, x^{(1)}) \quad (2.2)$$

- Chúng ta có thể thực hiện tóm tắt trích xuất hoặc trừu tượng hóa các văn bản bằng một mô hình ngôn ngữ phù hợp [72] [73][74]. Một hệ thống dịch máy có thể được phát triển đơn giản nếu chúng ta có các mô hình cho nhiều ngôn ngữ [75]. Trả lời câu hỏi là một trường hợp sử dụng ít rõ ràng hơn (có hoặc không có ngữ cảnh)[76]. Các mô hình ngôn ngữ cũng có thể được sử dụng để nhận dạng giọng nói, nhận dạng ký tự quang học (OCR), nhận dạng chữ viết tay và các ứng dụng khác [77]. Có nhiều khả năng [78].
- Chuỗi liên tục gồm n phần tử từ một mẫu văn bản hoặc âm thanh nhất định được định nghĩa là N-gram [79]. Tùy thuộc vào ứng dụng, các phần tử có thể là chữ cái, từ hoặc cặp cơ sở [80] [81]. N-gram thường được trích xuất từ một văn bản hoặc ngữ liệu âm thanh (Một tập dữ liệu văn bản dài). Một mô hình ngôn ngữ N-gram dự báo khả năng một N-gram cụ thể xuất hiện trong bất kỳ chuỗi từ nào trong ngôn ngữ
- [82] [83]. Một mô hình N-gram có năng lực có thể dự đoán từ tiếp theo trong câu, được biểu thị bằng giá trị $p(w|h)$.
- Ví dụ về N-gram bao gồm unigram ("This", "article", "is", "on", "NLP") và bigram ("This article", "article is", "is on", "on NLP").
- Có hai loại phương pháp mô hình hóa ngôn ngữ chính:
 -
 - • Mô hình ngôn ngữ thống kê
 - • Mô hình ngôn ngữ thần kinh

2.2.1 Statistical language models

Mô hình ngôn ngữ thống kê là cấu trúc của các mô hình xác suất có thể dự đoán từ tiếp theo trong một chuỗi dựa trên các từ đứng trước nó [84] [85]. Bên cạnh việc cung cấp xác suất cho mỗi chuỗi từ, các mô hình ngôn ngữ cũng cung cấp xác suất của một từ cụ thể (hoặc một chuỗi từ) theo sau một chuỗi từ. Một mô hình ngôn ngữ tìm hiểu khả năng xuất hiện của từ.

Các mô hình đơn giản hơn có thể xem xét ngữ cảnh của một chuỗi từ ngắn, nhưng các mô hình lớn hơn có thể xem xét các câu hoặc đoạn văn. Các mô hình ngôn ngữ thường được sử dụng nhất ở cấp độ từ. Tính toán xác suất n-gram tạo ra một mô hình ngôn ngữ xác suất cơ bản[86].

Khả năng của một n-gram là xác suất có điều kiện rằng từ của n-gram cuối cùng theo sau một n-1 gram nhất định. Trong thực tế, chính tỷ lệ xuất hiện của từ cuối cùng theo sau n-1 gram loại trừ từ cuối cùng. Đây là một giả định Markov: với n-1 gram (hiện tại), xác suất n-gram (tương lai) không phụ thuộc vào n-2, n-3, v.v. [87]. Chiến lược này có những nhược điểm rõ ràng. Quan trọng nhất là, phân phối xác suất của từ tiếp theo bị ảnh hưởng độc quyền bởi n từ trước đó. Các câu phức tạp bao gồm ngữ cảnh mở rộng, có thể có tác động đáng kể đến từ tiếp theo được chọn[88]. Do đó, ngay cả khi n lớn, từ tiếp theo

có thể không rõ ràng từ n từ trước đó.

Một cụm từ có thể tác động đến việc lựa chọn một từ trước đó: từ United có khả năng xảy ra cao hơn đáng kể nếu sau là States of America. Điều này được gọi là vấn đề ngữ cảnh. Ngoài ra, rõ ràng là chiến lược này không được mở rộng đúng cách: khi kích thước (n) tăng lên, số lượng các hoán vị tiềm năng tăng lên đáng kể, ngay cả khi phần lớn các biến thể không bao giờ xuất hiện trong văn bản. Và tất cả các xác suất xuất hiện (hoặc số lượng n-gram) phải được tính toán và lưu lại. Hơn nữa, các n-gram không xuất hiện gây ra vấn đề về tính thừa thót vì độ chi tiết của phân phối xác suất có thể tương đối thấp (xác suất từ có ít giá trị khác nhau, do đó hầu hết các từ đều có cùng xác suất).

2.2.2 Neural Language Models(NLM)

Việc sử dụng mạng no-ron trong việc xây dựng các mô hình ngôn ngữ gần đây đã tăng lên thành phương pháp thống trị và cách tiếp cận tốt nhất[89].[90] Mô hình hóa ngôn ngữ no-ron, hay viết tắt là NLM, là việc sử dụng mạng no-ron trong mô hình hóa ngôn ngữ. Đối với các nhiệm vụ khó như dịch máy[45], các kỹ thuật mạng no-ron hoạt động tốt hơn các phương pháp truyền thống trên cả các mô hình ngôn ngữ riêng lẻ và khi các mô hình được hợp nhất thành các mô hình lớn hơn [91]. Khả năng khai quát hóa của phương pháp này có thể là một yếu tố cơ bản cho bước nhảy vọt về hiệu suất tốt hơn. NLM đưa ra dự đoán về các mẫu phức tạp và trừu tượng trong ngôn ngữ và hoạt động tốt hơn các mô hình ngôn ngữ thông thường trong nhiều ứng dụng xử lý ngôn ngữ tự nhiên. NLM cũng đã được sử dụng trong các lĩnh vực như khử nhận dạng văn bản lâm sàng, tổng hợp phán đoán và phân tích cú pháp [92].

Mô hình ngôn ngữ no-ron (NLM) xử lý vấn đề về độ thừa thót dữ liệu n-gram bằng cách tham số hóa các từ dưới dạng vectơ (nhúng từ) và đưa chúng vào mạng no-ron[93]. Các tham số được khám phá trong suốt quá trình đào tạo. Các mô hình Ngôn ngữ no-ron tạo ra các nhúng từ có đặc tính là các từ gần nhau về mặt ngữ nghĩa và cũng gần nhau trong không gian vectơ cảm ứng.

Ba thuộc tính mô hình được liệt kê bên dưới có thể được sử dụng để xác định phương pháp mạng no-ron cho mô hình ngôn ngữ:

1. Gán một vectơ đặc trưng từ phân tán cho mỗi từ trong vốn từ vựng.
2. Về các vectơ đặc trưng của các từ trong chuỗi, hãy biểu thị hàm xác suất chung của các chuỗi từ.
3. Tìm hiểu cả vectơ đặc trưng từ và các tham số hàm xác suất cùng một lúc.

Đây là một mô hình khá đơn giản trong đó biểu diễn và mô hình xác suất đều được học trực tiếp từ đầu vào văn bản thô. Các kỹ thuật dựa trên no-ron vượt trội hơn các phương pháp thống kê truyền thống. Có hai khái niệm ở đây. Hai khái niệm này là Mô hình ngôn ngữ thông thường và Mô hình ngôn ngữ có mặt nạ. Mô hình ngôn ngữ nhân quả gọi ý dự đoán mã thông báo tiếp theo trong một loạt mã thông báo.

Mặt khác, mã thông báo có mặt nạ trong một chuỗi được dự đoán bằng mô hình ngôn ngữ có mặt nạ và mô hình có thể giữ các mã thông báo theo hai hướng[94] [95]. Cách các mô hình ngôn ngữ dựa trên mạng nơ-ron mã hóa đầu vào làm giảm bớt vấn đề thừa thót. Nếu bạn không quen với nhúng từ, các lớp nhúng tạo ra một vectơ có kích thước tùy ý của mỗi từ bao gồm các liên kết ngữ nghĩa. Các vectơ liên tục này cung cấp độ chi tiết cho phân phối xác suất của từ tiếp theo. Hơn nữa, vì mô hình ngôn ngữ thực sự là một hàm (cũng như tất cả các mạng nơ-ron có nhiều phép tính ma trận), nên không cần phải duy trì tất cả các số đếm n-gram để tạo ra phân phối xác suất của từ tiếp theo.

2.3 Sequence Models

Nghiên cứu về dữ liệu tuần tự như câu văn bản và dữ liệu chuỗi thời gian đã dẫn đến sự phát triển của Mô hình trình tự. Các mô hình này được xây dựng riêng để xử lý dữ liệu tuần tự. Mô hình trình tự thường được áp dụng trong nhiều ứng dụng khác nhau, bao gồm NLP.

2.3.1 Recurrent Neural Network (RNN)

Mạng nơ-ron hồi quy (RNN) là một loại thiết kế mạng nơ-ron được sử dụng để khám phá các mẫu trong chuỗi dữ liệu chuỗi thời gian. RNN được sử dụng ở cấp độ ứng dụng cao hơn như mô hình ngôn ngữ và tạo văn bản [96]. RNN có “trạng thái nội bộ” được cập nhật khi chuỗi được xử lý như thể hiện trong Hình.2.2.

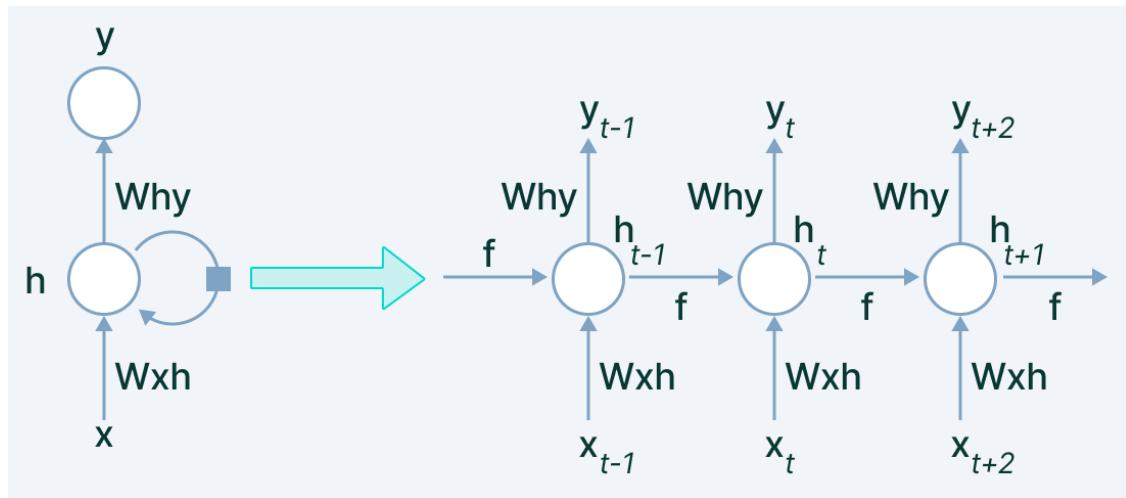


Figure 2.2: Recurrent Neural Network(RNN) [97]

Cách thức thông tin được vận chuyển qua mạng phân biệt Mạng nơ-ron hồi quy với Mạng nơ-ron truyền thẳng, thường được gọi là Perceptron đa lớp (MLP)[29].

Trong khi Mạng truyền thẳng gửi dữ liệu qua mạng mà không sử dụng chu

kỳ, RNN sử dụng chu kỳ và gửi dữ liệu trở lại chính nó như thể hiện trong Hình 2.3. Điều này cho phép chúng mở rộng khả năng của Mạng truyền thẳng để bao gồm các đầu vào trước đó ngoài đầu vào hiện tại. RNN có trạng thái ẩn cho phép sử dụng lại các đầu ra trước đó làm đầu vào. Nhưng RNN có vấn đề về độ dốc biến mất, khiến việc học các chuỗi dữ liệu lớn trở nên khó khăn. Ngoài ra, nó không thể xử lý các chuỗi rất dài do cách xử lý tuần tự[98].

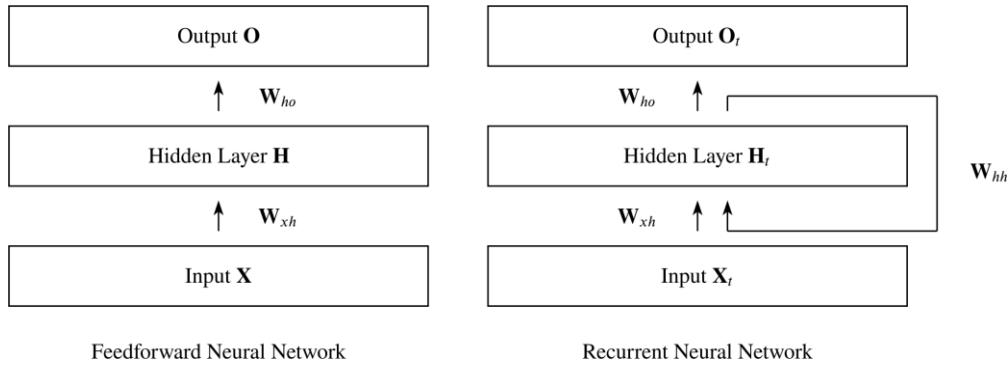


Figure 2.3: Feedforward Neural Network and Recurrent Neural Network

2.3.2 LSTM

Mạng bộ nhớ dài hạn ngắn hạn (LSTM) là phiên bản sửa đổi của RNN giúp cải thiện khả năng nhớ lại bộ nhớ. Vấn đề độ dốc biến mất của RNN được LSTM giải quyết bằng cách thêm các cổng 'quên'. Là một giải pháp cho bộ nhớ ngắn hạn, LSTM và GRU đã được phát triển [99] [100]. Như thể hiện trong Hình 2.4. LSTM chứa các hệ thống nội bộ được gọi là các cổng, cuối cùng cho phép kiểm soát luồng thông tin. Các cổng này có thể tìm hiểu dữ liệu nào trong một chuỗi nên được giữ lại hoặc loại bỏ. Điều này cho phép nó truyền tải thông tin quan trọng xuống chuỗi trình tự dài để tạo ra các dự đoán [101].

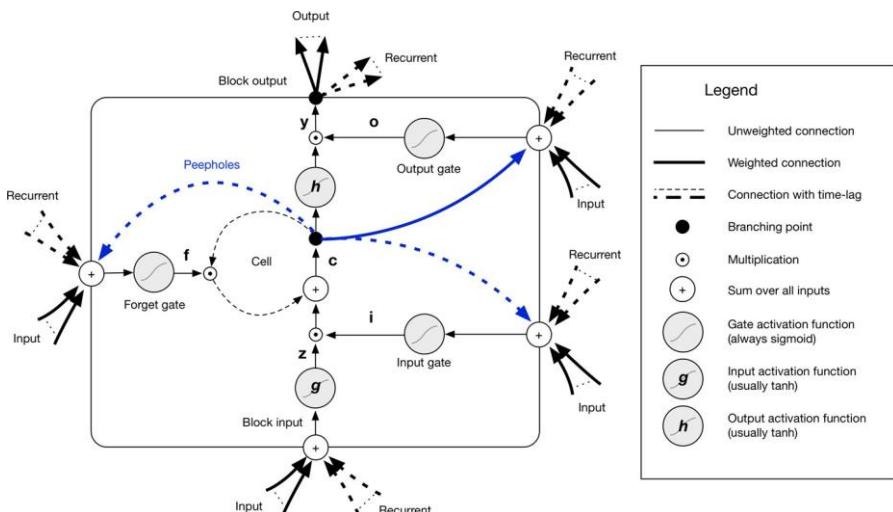


Figure 2.4: LSTM [40]

2.3.3 Transformers

Transformers [30] là một loại mô hình mạng nơ-ron tiên tiến được sử dụng trong xử lý ngôn ngữ tự nhiên cho phép song song hóa quá trình xử lý và dựa trên cơ chế tự chú ý thay vì mạng nơ-ron hồi quy. Mô hình transformer được hiển thị trong Hình.2.5 được giới thiệu vào năm 2017 để giải quyết vấn đề biến mất gradient và hoạt động theo cách song song, do đó, nó vượt trội hơn hiệu suất của RNN. Ngoài ra, nó có cơ chế chú ý nhiều đầu như được hiển thị trong Hình.2.6.

Transformers chứa một số lượng lớn các mô hình được đào tạo trước có thể được sử dụng cho nhiều tác vụ và tập dữ liệu khác nhau. Transformers đã chứng minh rằng chúng có thể là trình học đa nhiệm ít lần [31] và không giám sát [32].

Transformers chứng minh rằng chúng có thể được áp dụng cho bất kỳ tác vụ đường ống nào như dịch máy, tạo văn bản thành văn bản, phân loại và các tác vụ khác. Phương trình 2.3 được sử dụng để tính toán sự chú ý tích vô hướng được chia tỷ lệ

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2.3)$$

where $\sqrt{d_k}$ is the dimension of the key vector k and query vector q.

Để thể hiện sự chú ý đa hướng, nó được nêu trong Công thức 2.4, trong đó, hướng của mỗi sự chú ý được thể hiện trong Công thức 2.5.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O \quad (2.4)$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \quad (2.5)$$

Các mô hình dựa trên Transformers [30], chẳng hạn như RoBERTa [102], GPT-2 [32] và MarianMT đã chứng minh là cực kỳ hiệu quả đối với nhiều tác vụ NLP và là cốt lõi của các tác vụ NLP đối với nhiều loại nghiên cứu. Khả năng thích ứng và phục hồi của Transformers là những lý do chính khiến chúng được chấp nhận rộng rãi. Transformers đã được chứng minh là vượt trội hơn nhiều so với các mô hình trình tự trước đó như RNN và LSTM[103].

Đổi mới cơ bản của Transformers là cơ chế tự chú ý bỏ qua các hạn chế của RNN bằng cách cho phép mỗi mã thông báo trong trình tự đều vào chú ý đến mọi mã thông báo khác trong trình tự một cách độc lập.

Tự chú ý được xử lý song song cho mỗi mã thông báo của trình tự đều vào, tránh sự phụ thuộc tuẫn tự được tìm thấy trong các mạng nơ-ron hồi quy và LSTM[104]. Tính song song này cho phép Transformers tận dụng toàn bộ sức mạnh của máy tính hiện tại. OpenAI và Google đã phát hành các mô hình ngôn ngữ được đào tạo trước dựa trên Transformers chỉ chứa kiến trúc bộ mã hóa Transformers như BERT [60].

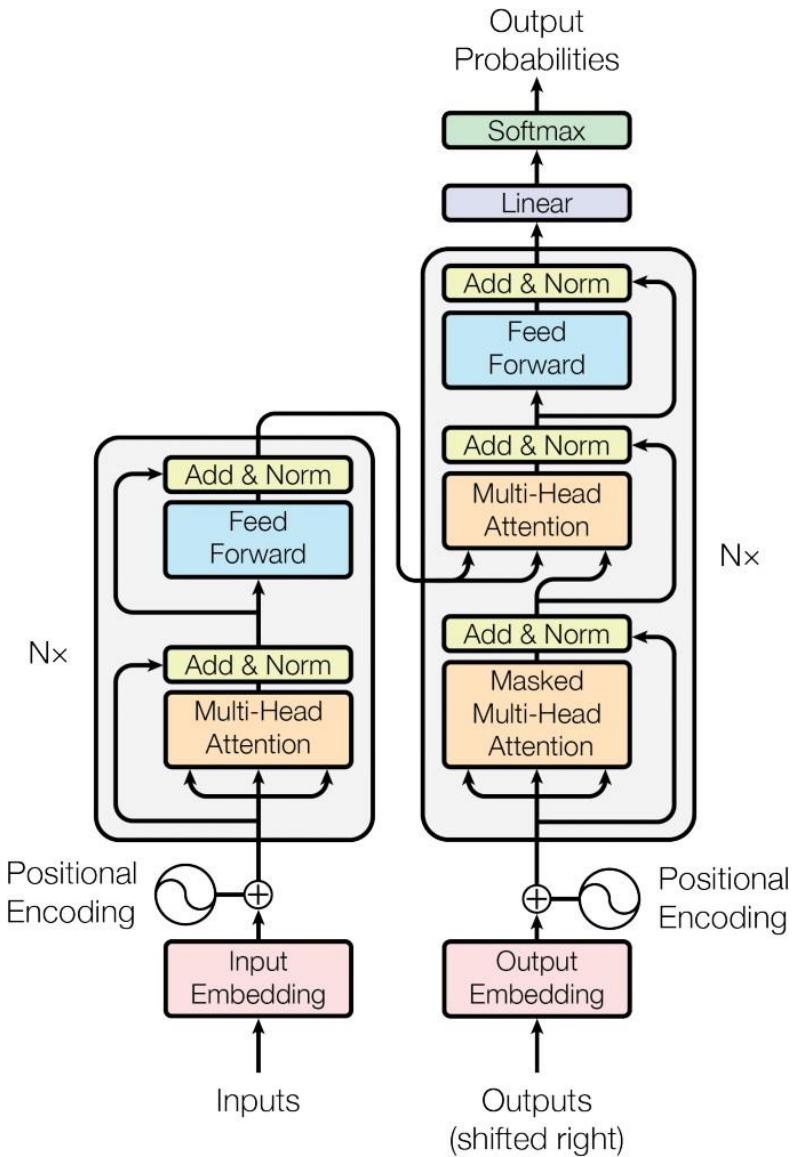


Figure 2.5: Transformer Architecture [30]

Mặt khác, OpenAI và Google cũng đã phát hành GPT [59] là kiến trúc bộ giải mã biến áp duy nhất. BERT và GPT đã được chứng minh là yêu cầu tính chỉnh tương đối ít sau khi được đào tạo trước để phá vỡ các kết quả tiên tiến trên hơn một chục tác vụ NLU. Các mô hình biến áp có ba kiến trúc khác nhau. Loại đầu tiên là các biến áp chỉ có bộ mã hóa như RoBERTa, Electra và Luke. Loại biến áp thứ hai là biến áp chỉ có bộ giải mã, chẳng hạn như GPT-2. Cuối cùng, các mô hình biến áp bộ mã hóa-giải mã như MarianMT, T5 và BART được sử dụng. Đối với các mô hình biến áp bộ mã hóa-giải mã, bộ mã hóa nhận một chuỗi đầu vào và xây dựng một biểu diễn trung gian thông qua các cơ chế nhúng và chú ý. Bộ giải mã nhận được biểu diễn trung gian hoặc trạng thái ẩn và bắt đầu tạo một chuỗi đầu ra [105].

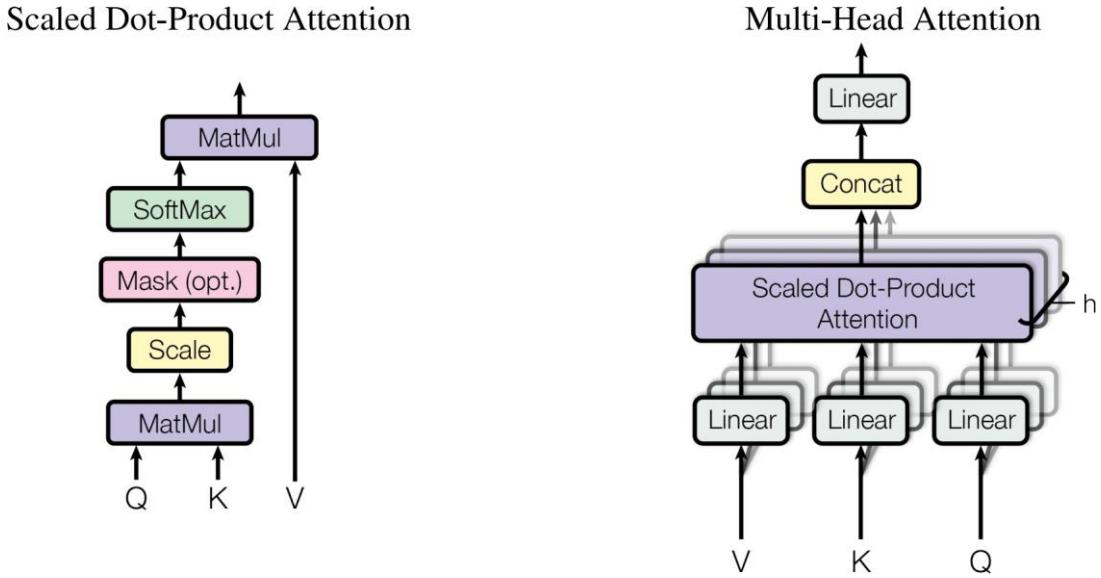


Figure 2.6: Attention Mechanism - (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel [30]

2.3.4 The evolution of language models

Mặc dù mạng nơ-ron giải quyết được vấn đề thưa thót, nhưng vấn đề ngữ cảnh vẫn tồn tại. Các mô hình ngôn ngữ được xây dựng theo cách giải quyết được vấn đề ngữ cảnh và trở nên hiệu quả hơn — đưa các từ ngữ cảnh ngày càng tăng vào ảnh hưởng đến phân phối xác suất và thực hiện điều đó hiệu quả hơn [106]. Ngoài ra, mục tiêu của các mô hình ngôn ngữ là thiết kế một kiến trúc cho phép mô hình tìm hiểu các thuật ngữ ngữ cảnh nào quan trọng hơn các thuật ngữ ngữ cảnh khác.

Sự phát triển của các mô hình ngôn ngữ để cập đến sự phát triển liên tục của các mô hình ngày càng tinh vi, từ các mô hình n-gram đơn giản đến các mô hình mạng nơ-ron tiên tiến, có khả năng dự đoán chính xác và tạo ra ngôn ngữ giống con người, khiến chúng trở thành công cụ quan trọng trong nhiều ứng dụng xử lý ngôn ngữ tự nhiên [107].

Việc sử dụng Mạng nơ-ron hồi quy (RNN) là một bổ sung trong lĩnh vực này. Vì nó là mạng dựa trên ô LSTM hoặc GRU, nên nó sẽ xem xét tất cả các từ trước đó trong khi chọn từ tiếp theo. Để biết thêm thông tin về cách RNN đạt được bộ nhớ mở rộng, AllenNLP đã xuất Nhúng cho Mô hình Ngôn ngữ (ELMo)

[58] tăng cường ý tưởng này bằng cách sử dụng LSTM hai chiều như thể hiện trong Hình.2.7, cho phép xem xét tất cả ngữ cảnh trước và sau từ [108].

Nhược điểm cơ bản của các thiết kế dựa trên RNN là bản chất tuần tự của chúng. Do đó, thời gian đào tạo cho các chuỗi lớn tăng vọt do thiếu song song hóa. Kiến trúc biến áp là câu trả lời cho tình thế tiến thoái lưỡng nan này [30].

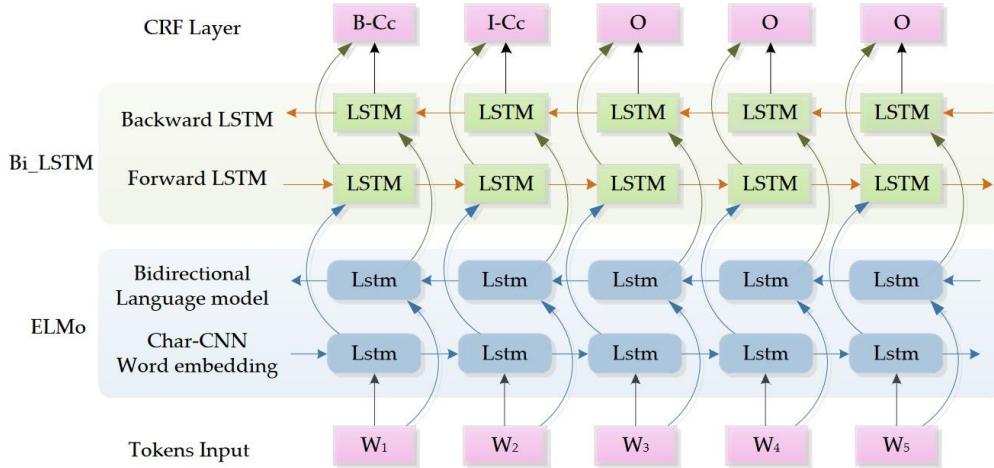


Figure 2.7: The architecture of embeddings from language models (ELMO)-based BiLSTM-CRF model [108]

Thiết kế máy biến áp cũng được tối ưu hóa để tạo mô hình GPT của OpenAI và mô hình BERT từ Google. Các mô hình đó cũng sử dụng một quy trình được gọi là Attention, cho phép mô hình tìm hiểu đâu vào nào đáng được chú ý hơn những đâu vào khác trong các tình huống cụ thể.

Những tiến bộ lượng tử quan trọng trong thiết kế mô hình ban đầu là RNN (đặc biệt là LSTM và GRU), giải quyết được vấn đề thưa thớt và cho phép các mô hình ngôn ngữ sử dụng ít không gian đĩa hơn đáng kể, sau đó là kiến trúc máy biến áp, cho phép song song hóa và tạo ra các cơ chế chú ý[109].

Các mô hình ngôn ngữ trước đây được sử dụng cho các tác vụ NLP thông thường như gắn thẻ POS hoặc dịch máy với các điều chỉnh nhỏ. Ví dụ, BERT có thể là một máy gắn thẻ POS với một chút đào tạo lại do khả năng trùu tượng của nó trong việc hiểu cấu trúc cơ bản của ngôn ngữ nói. Nay giờ, chúng ta có thể đào tạo các mô hình ngôn ngữ cho nhiều ứng dụng và cho nhiều mục đích, chẳng hạn như mô hình T5 [105] từ Google, như minh họa trong Hình 2.8.

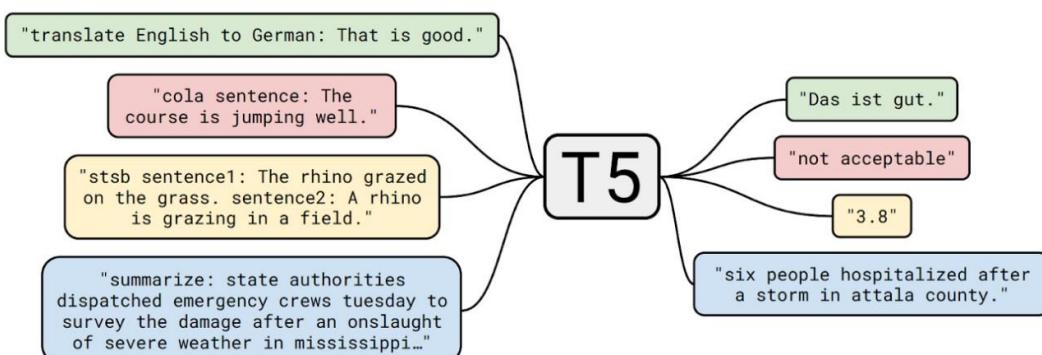


Figure 2.8: T5 Model capabilities [110]

2.4 Transformer-Based Processing

Xử lý dựa trên Transformer là một loại kiến trúc mạng nơ-ron đang ngày càng phổ biến trong những năm gần đây. Nó được sử dụng trong nhiều ứng dụng [43] [111] [112]

[113] như dịch ngôn ngữ máy, chatbot đàm thoại và thậm chí để cung cấp năng lượng cho các công cụ tìm kiếm tốt hơn. Kiến trúc Transformer là khối xây dựng cơ bản của tất cả các Mô hình ngôn ngữ có Transformer (LLM) [67]. Mạng nơ-ron Transformer lần đầu tiên được đề xuất trong một bài báo năm 2017 để giải quyết một số vấn đề của RNN đơn giản. Ý tưởng đằng sau Transformer là sử dụng sự chú ý của bản thân để tăng tốc độ mô hình có thể dịch từ chuỗi này sang chuỗi khác.

Để hiểu Transformer, trước tiên người ta phải hiểu cơ chế chú ý, cho phép Transformer có bộ nhớ cực kỳ dài hạn. Trong cơ chế này, mô hình tập trung vào các phần khác nhau của chuỗi đầu vào khi tạo từng mã thông báo đầu ra.

Về bản chất, kiến trúc Transformer sử dụng một cơ chế được gọi là sự chú ý của bản thân để xử lý đồng thời các chuỗi dữ liệu, chẳng hạn như câu văn bản hoặc thông tin chuỗi thời gian, thay vì dựa vào một quy trình tuần tự.

Khả năng xử lý song song này nâng cao hiệu quả và hiệu suất xử lý chuỗi, khiến nó có thể áp dụng cao cho nhiều tác vụ khác nhau vượt ra ngoài phạm vi NLP. Sau đây là một số thành phần và khái niệm chính liên quan đến xử lý dựa trên Transformer:

- **Tự chú ý:** Tự chú ý là một cơ chế cho phép mỗi phần tử trong chuỗi tập trung vào các phần tử khác trong cùng chuỗi, gán các mức độ quan trọng khác nhau cho từng phần tử. Cơ chế này nắm bắt các mối quan hệ và sự phụ thuộc giữa các phần tử trong chuỗi, điều này rất quan trọng để hiểu ngữ cảnh trong ngôn ngữ tự nhiên và dữ liệu tuần tự khác.
- **Tự chú ý nhiều đầu:** Trong kiến trúc Transformer, tự chú ý được áp dụng nhiều lần song song, mỗi lần có các tập hợp trọng số đã học khác nhau. Điều này cho phép mô hình tập trung vào các khía cạnh khác nhau của chuỗi đầu vào cùng một lúc, cho phép nó nắm bắt cả sự phụ thuộc cục bộ và toàn cục.
- **Mã hóa vị trí:** Transformer không có ý thức tích hợp về thứ tự hoặc vị trí trong chuỗi vì tự chú ý hoạt động trên các phần tử một cách độc lập. Để giải quyết vấn đề này, các mã hóa vị trí được thêm vào nhúng đầu vào để cung cấp cho mô hình thông tin về vị trí của từng phần tử trong chuỗi.
- **Kiến trúc mã hóa-giải mã:** Bộ chuyển đổi thường được sử dụng trong kiến trúc mã hóa-giải mã cho các tác vụ như dịch máy và tóm tắt văn bản. Bộ mã hóa xử lý chuỗi đầu vào, trong khi bộ giải mã tạo chuỗi đầu ra. Kiến trúc này cực kỳ hiệu quả đối với các tác vụ chuỗi-sang-chuỗi.
- **Mô hình được đào tạo trước:** Bộ chuyển đổi có thể được đào tạo trước trên các tập hợp lớn văn bản hoặc dữ liệu khác và các mô hình được đào tạo trước này có thể được tinh chỉnh cho các tác vụ cụ thể. Các mô hình được đào tạo trước, chẳng hạn như BERT, GPT và RoBERTa, đã đạt được kết quả tiên tiến trên nhiều tác vụ NLP và hơn thế nữa.
- **Học chuyển giao:** Học chuyển giao là một phương pháp phổ biến với Bộ chuyển đổi được đào tạo trước. Các mô hình được đào tạo trước trên các tập dữ liệu lớn có thể được tinh chỉnh trên các tập dữ liệu nhỏ hơn, dành riêng cho tác vụ, cho phép chúng thích ứng với các tác vụ cụ thể với tương đối ít đào tạo bổ

sung.

- Ứng dụng: Xử lý dựa trên Transformer đã được áp dụng cho nhiều tác vụ khác nhau, bao gồm dịch máy, tạo văn bản, phân tích tình cảm, trả lời câu hỏi, chú thích hình ảnh, hệ thống đề xuất, v.v. Khả năng xử lý dữ liệu tuần tự hiệu quả và hiệu suất cao đã khiến chúng trở thành lựa chọn hàng đầu của nhiều học viên học máy.

Tóm lại, xử lý dựa trên Transformer tận dụng cơ chế tự chú ý và khả năng xử lý song song của kiến trúc Transformer để phân tích và tạo chuỗi dữ liệu. Các mô hình này đã đạt được thành công đáng kể trong nhiều ứng dụng và tiếp tục thúc đẩy những tiến bộ trong xử lý ngôn ngữ tự nhiên và các tác vụ dữ liệu tuần tự khác.

2.5 Transfer Learning and Knowledge Distillation

2.5.1 Pre-trained Language Models

Ngày nay, các mô hình ngôn ngữ được đào tạo trước đã chứng kiến thành công to lớn trong lĩnh vực NLP [66]. Một mô hình được đào tạo trước là một mô hình đã được đào tạo trên một tập dữ liệu chuẩn lớn để giải quyết một số vấn đề và sau đó lưu mạng này với các trọng số để được đào tạo và sử dụng lại cho một tác vụ khác. Các mô hình được đào tạo trước thường được sử dụng làm cốt lõi của công việc học chuyển giao [114]. Có nhiều mô hình ngôn ngữ với số lượng tham số khác nhau [115] và được lấy từ Transformers và các mô hình trước đó được đào tạo trước khác như thể hiện trong Hình 2.9.

Thông qua đào tạo trước và tinh chỉnh, chúng ta có thể tăng cường độ mạnh mẽ và độ không chắc chắn của mô hình. Có một số phương pháp [116] [117] [118] [119] [120] [121] cho phép các mô hình ngôn ngữ được đào tạo trước đào tạo các mô hình lớn với hàng tỷ tham số từ các tập hợp dữ liệu không có nhãn quy mô lớn theo cách tự giám sát. Nghiên cứu gần đây [122] [123]

[66] [124] [125] [126] đã chỉ ra việc sử dụng các mô hình được đào tạo trước và cũng chứng minh những lợi ích của việc sử dụng các mô hình ngôn ngữ được đào tạo trước cho nhiều tác vụ như dịch máy. Với các nguồn tài nguyên điện toán hạn chế, có thể đào tạo các mô hình dịch thuật có khả năng cạnh tranh với các mô hình tiên tiến. Việc sử dụng một mô hình ngôn ngữ được đào tạo trước với cùng một kiến trúc từ tác vụ này sang tác vụ khác và thích ứng với tác vụ khác là một bước quan trọng hướng tới việc phát triển một mô hình mới đáng tin cậy và hiệu quả.

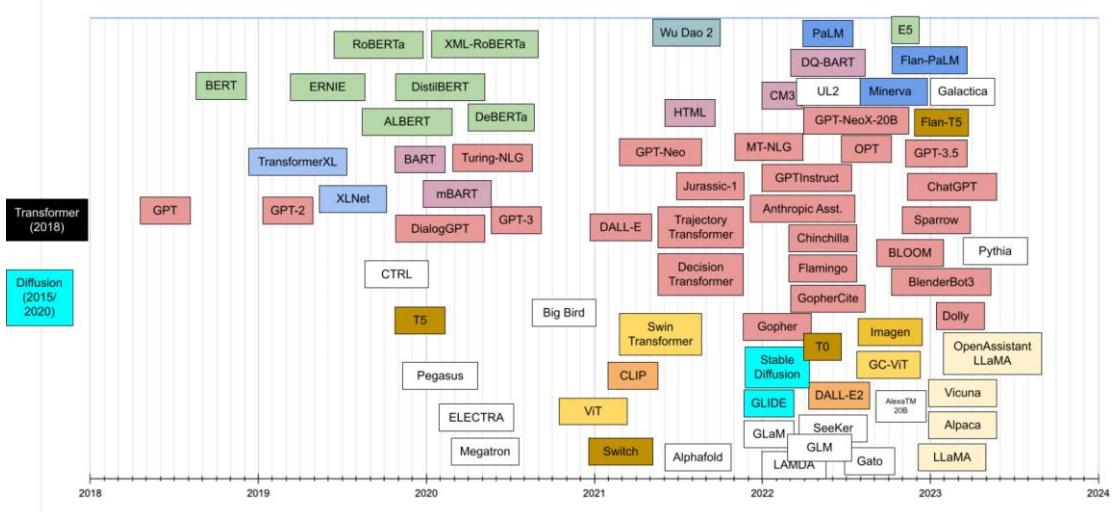


Figure 2.9: Transformer timeline and Pre-trained language models . Colors describe Transformer family [115]

mô hình. Tiền đào tạo đã dẫn đến những cải tiến đáng kể trong cả các tác vụ hạ nguồn chế độ dữ liệu thấp và các công việc có đủ dữ liệu và do đó là lý do thúc đẩy sự phổ biến của các máy biến áp trong NLP hiện đại. Tiền đào tạo Các máy biến áp đang hoạt động theo cách tự giám sát và chúng được đào tạo trên các tập đoàn dữ liệu lớn cho thấy thành công lớn khi được tinh chỉnh trên các tác vụ NLP hạ nguồn bao gồm dịch máy, trả lời câu hỏi và tóm tắt văn bản.

2.5.2 Transfer Learning

Học chuyển giao là một phương pháp học máy trong đó một mô hình được đào tạo cho một công việc và sau đó được sử dụng lại cho một tác vụ định hướng khác [127] [110] như thể hiện trong Hình 2.10. Vì vậy, thay vì tạo và đào tạo một mô hình từ đầu, tốn kém, mất thời gian và đòi hỏi khối lượng dữ liệu lớn, bạn có thể chỉ cần tinh chỉnh một mô hình được đào tạo trước như BERT để điều chỉnh nhiều tác vụ như thể hiện trong Hình 2.11. Điều này có nghĩa là các tổ chức có thể thực hiện các tác vụ NLP nhanh hơn và với ít dữ liệu được gắn nhãn hơn.

2.5.3 Knowledge Distillation

Chắt lọc kiến thức là một chiến lược đào tạo sử dụng chuyển giao kiến thức để đào tạo các mô hình nhỏ để có độ chính xác như các mô hình lớn hơn [128] [129]. Như thể hiện trong Hình 2.12, mô hình lớn hơn được gọi là "mạng giáo viên" trong lĩnh vực chắt lọc kiến thức, trong khi mạng nhỏ hơn được gọi là "mạng học sinh".

Mô hình giáo viên có thể là một mô hình lớn duy nhất hoặc một nhóm các mô hình độc lập đã được đào tạo trên các nguồn dữ liệu phức tạp như dữ liệu có độ phân giải thấp, dữ liệu đa miền và đa tác vụ hoặc dữ liệu liên phương thức [129].

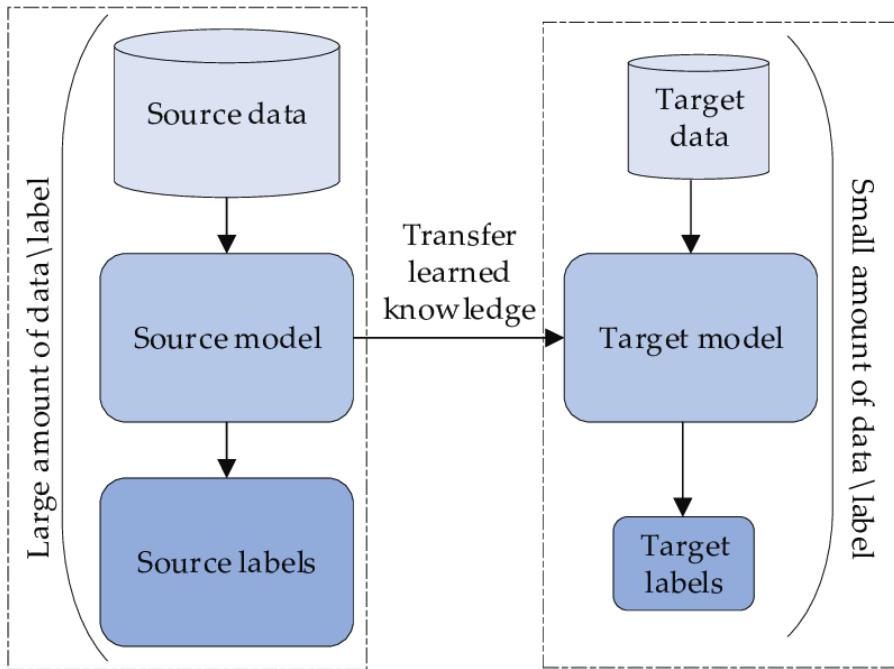


Figure 2.10: Transfer learning and fine-tuning pre-trained models [127]

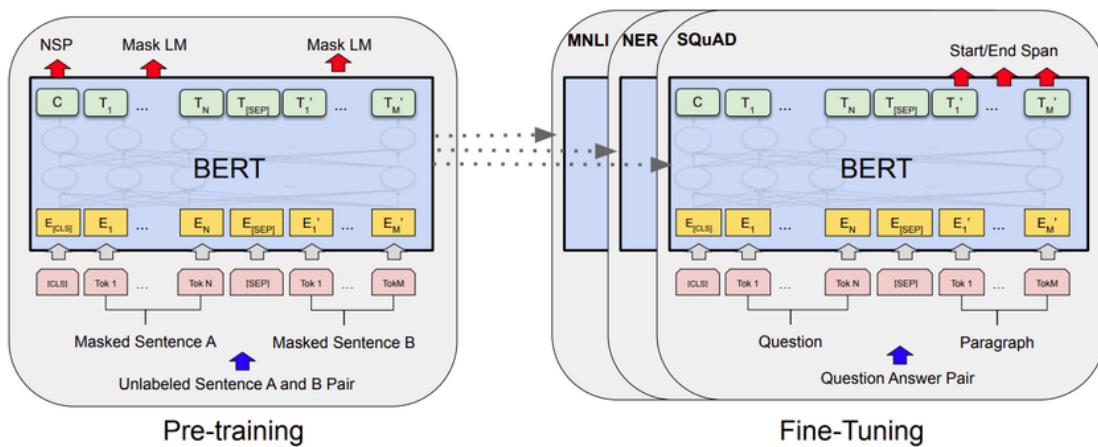


Figure 2.11: Fine-tuning pre-trained BERT language model [60]

Sau khi đào tạo mô hình giáo viên, quá trình chưng cất có thể thu thập và chuyển thông tin đã đào tạo sang mô hình học sinh nhỏ hơn, sau đó đào tạo mạng học sinh. Điều này có thể được thực hiện thông qua nhiều mối quan hệ khác nhau giữa các mạng giáo viên-học sinh như thể hiện trong Hình 2.13. Xử lý ngôn ngữ tự nhiên, nhận dạng giọng nói, nhận dạng hình ảnh và hệ thống đề xuất đều được hưởng lợi từ phương pháp nén mô hình chưng cất kiến thức [130]. Hơn nữa, để bảo mật quyền riêng tư dữ liệu, chưng cất kiến thức cung cấp các mạng nén mô hình hỗ trợ quyền riêng tư.

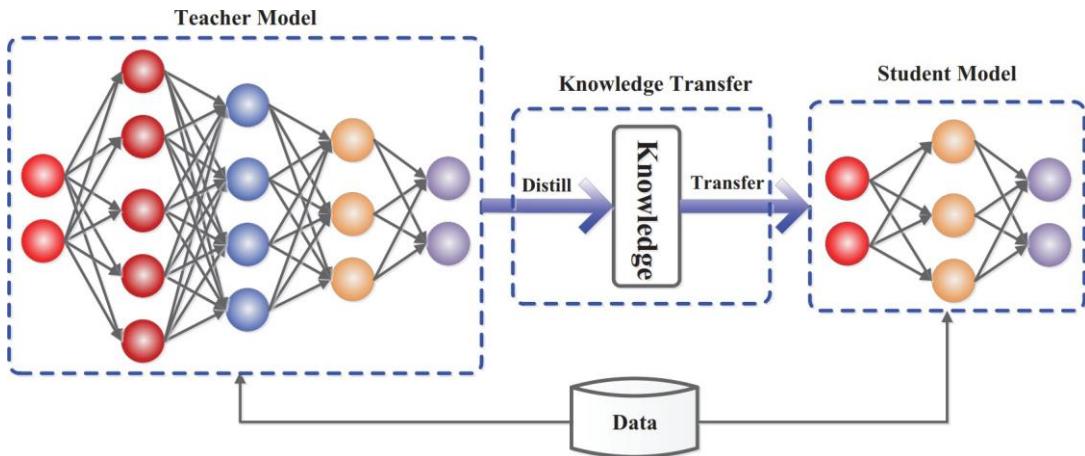


Figure 2.12: Knowledge Distillation [129]

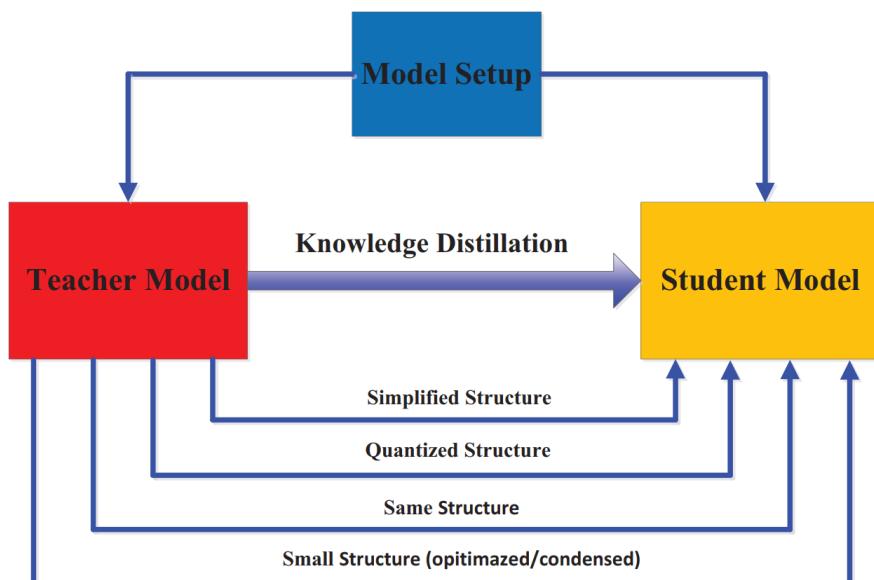


Figure 2.13: Various relationships among teacher-student networks [129]

2.6 Evaluation metrics of seq2seq models

Các mô hình Seq2seq, còn được gọi là các mô hình mã hóa-giải mã, đã được sử dụng rộng rãi trong nhiều tác vụ xử lý ngôn ngữ tự nhiên như dịch ngôn ngữ, tóm tắt văn bản, trả lời câu hỏi và chatbot đàm thoại. Đánh giá hiệu suất của các mô hình Seq2seq là chìa khóa để xác định độ chính xác và hiệu quả của chúng trong các tác vụ này.

Các số liệu đánh giá chung của các mô hình Seq2seq bao gồm độ phức tạp, điểm BLEU và điểm ROUGE, tất cả đều cung cấp thông tin chi tiết về chất lượng đầu ra được tạo ra so với văn bản tham chiếu hoặc văn bản chuẩn vàng. Do đó, việc hiểu và áp dụng đúng các số liệu này là rất quan trọng để đo lường hiệu suất của các mô hình Seq2seq một cách hiệu quả.

Hình 2.14 cho thấy có những đầu vào nguồn có thể được người dùng xử lý và văn bản tham chiếu là văn bản gốc không được xử lý. Có thể đo lường việc so sánh văn bản tham chiếu với văn bản được tạo ra bằng một số số liệu để có được điểm khớp. Có một

số kỹ thuật hoặc thuật toán cho dịch máy như điểm BLEU, SacreBLEU và điểm ROUGE.

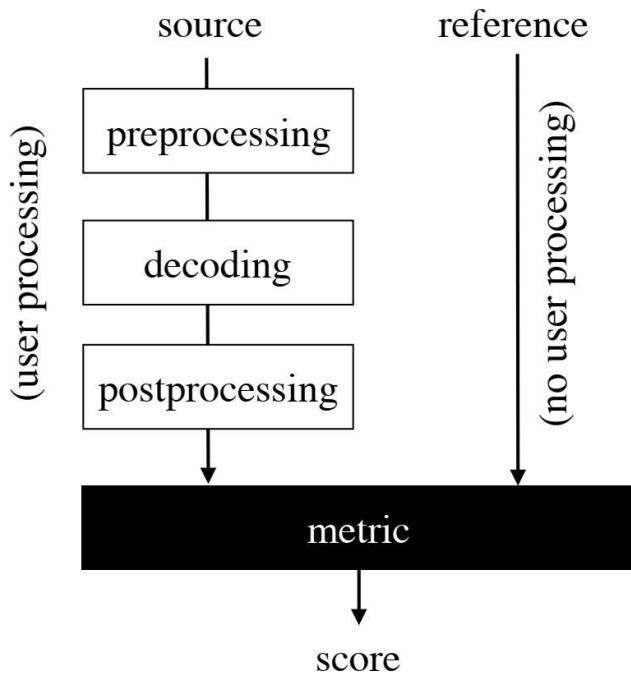


Figure 2.14: The proper pipeline for computing reported matching scores

2.6.1 BLEU Score

Đánh giá thủ công kết quả dịch máy tốn nhiều thời gian và chi phí. Đánh giá của con người có thể mất vài ngày hoặc vài tuần để hoàn thành, do đó, một hệ thống tính điểm mới đã được tạo ra để tự động hóa quy trình đánh giá. Phương pháp này thường được gọi là Điểm BLEU [131] (Sinh viên đánh giá song ngữ). BLEU là điểm số đo lường được sử dụng để so sánh bản dịch văn bản của ứng viên với một hoặc nhiều bản dịch tham chiếu. Mặc dù được thiết kế để dịch, nhưng nó cũng có thể được sử dụng để đánh giá đầu ra văn bản cho nhiều công việc xử lý ngôn ngữ tự nhiên.

Thuật toán BLEU đánh giá chất lượng văn bản dịch máy từ ngôn ngữ này sang ngôn ngữ khác. Đây là phương pháp tự động được sử dụng rộng rãi nhất để đo lường chất lượng dịch máy[132].

Mỗi quan hệ giữa kết quả của máy và kết quả của một người được cho là chỉ là chất lượng. Bản dịch máy càng gần với bản dịch của con người chuyên nghiệp thì càng tốt. Đây là khái niệm cốt lõi đằng sau BLEU. BLEU dễ sử dụng, dễ chạy, không phụ thuộc vào ngôn ngữ và có mối tương quan chặt chẽ với đánh giá của con người.

Một mối tương quan mạnh nhận được điểm là 1.0, trong khi một sự không khớp hoàn toàn nhận được điểm là 0.0. BLEU là một trong những biện pháp đầu tiên khẳng định có mối liên hệ chặt chẽ với các đánh giá chất lượng của con người và vẫn là một trong những số liệu đánh giá tự động và chi phí thấp phổ biến nhất.

Biện pháp BLEU gán điểm dịch từ 0 đến 1, tuy nhiên, nó thường được thể hiện dưới dạng giá trị phần trăm, như minh họa trong Bảng 2.1. Bản dịch càng gần 1 thì càng giống với bản dịch của con người.

Table 2.1: BLEU score

BLEU Score	Interpretation
Less than 10	Almost useless
10 - 19	Hard to get the gist
20 - 29	The gist is clear, but has significant grammatical errors
30 - 40	Understandable to good translations
40 - 50	High quality translations
50 - 60	Very high quality, adequate, and fluent translations
Greater than 60	Quality often better than human

- Nói một cách đơn giản, thống kê BLEU đếm số lượng từ trùng nhau trong một bản dịch nhất định khi so sánh với bản dịch tham chiếu, với các từ liên tiếp được chấm điểm cao hơn. BLEU tính điểm của mình dựa trên n-gram khớp trong các văn bản mà nó so sánh. N-gram là một chuỗi "n" mục từ một văn bản trong NLP. Các thành phần này có thể bao gồm các từ, chữ cái, v.v.
- Văn bản đã dịch (bản dịch ứng viên) được so sánh với bản dịch tham chiếu và n-gram khớp được đếm. Tuy nhiên, điểm số BLEU NLP thay đổi số lượng ước tính để đảm bảo tính chính xác, một quy trình được gọi là độ chính xác n-gram đã sửa đổi.
- BLEU sử dụng một phương pháp thống kê áp dụng hình phạt ngắn gọn (BP) cho độ chính xác đã sửa đổi. Hình phạt về độ ngắn gọn được tính như sau:
-
- • Chỉ khi số từ trong bản dịch ứng viên vượt quá số từ trong bản dịch tham khảo thì hình phạt về độ ngắn gọn mới bằng 1.
- • Nếu số từ trong văn bản ứng viên ít hơn hoặc bằng số từ trong văn bản tham khảo thì hình phạt về độ ngắn gọn được tính theo Công thức 2.6.

$$BP = e^{-c} \quad (2.6)$$

trong đó c là tổng số unigram (chiều dài) trong tất cả các câu ứng viên và r là chiều dài đánh giá cao nhất trong ngữ liệu cho mỗi câu ứng viên. Điểm BLEU được tính bằng Công thức 2.7.

$$BLEU = BP \cdot \exp \sum_{n=1}^N w_n \cdot \log p_n \quad (2.7)$$

Số lượng n-gram khớp là quan trọng trong công thức toán học này. Khi một kết quả khớp 1-gram được xác định, phép cộng sẽ bắt đầu, tức là khi $n = 1$. Và cứ thế cho đến khi đạt đến tổng số kết quả khớp (N). P_n là giá trị độ chính xác cho mỗi n-gram khớp; do đó, $n = 1$ tương ứng với P_1 .

SacreBLEU [133] là phần mềm python nhằm mục đích tôn trọng BLEU hơn. Nó yêu cầu các đầu ra được tách mã và tạo ra các kết quả giống như WMT bằng cách sử dụng tiền xử lý nội bộ số liệu của riêng nó. Hơn nữa, nó tạo ra một chuỗi phiên bản ngắn gọn mô tả các cài đặt đã được áp dụng.

2.6.2 Exact Match Accuracy

Biện pháp này khá đơn giản để tính toán. Thước đo này được sử dụng để so sánh điểm giống và khác nhau giữa hai văn bản. Eq.2.8 cho thấy cách tính toán và đo lường thước đo này cho hai câu $y(i)$ và $\hat{y}^{(i)}$. Câu đầu tiên $y(i)$ là câu tham chiếu, chứa 100% câu thực sự cần thiết. Câu thứ hai $\hat{y}^{(i)}$, là câu dự đoán được tạo ra bởi mô hình.

$$ExactMatchAccuracy = \frac{1}{n} \sum_{i=1}^n [I(y^{(i)} == \hat{y}^{(i)})] \quad (2.8)$$

trong đó n là số ví dụ, $y(i)$ là nhãn thực cho ví dụ thứ i trong văn bản tham chiếu và $\hat{y}^{(i)}$ là nhãn dự đoán cho ví dụ thứ i .

Ví dụ về cách chấm điểm Độ chính xác khớp chính xác của hai câu được thể hiện trong Hình.2.15. Độ chính xác khớp chính xác = 1 nếu các ký tự trong dự đoán của mô hình hoàn toàn khớp với tất cả các ký tự của phản hồi tham chiếu chính hằng; nếu không, Độ chính xác khớp chính xác được xác định bằng cách sử dụng các ký tự có thể so sánh được và nằm trong khoảng từ 0 đến 1. Độ chính xác khớp chính xác bằng 0 nếu tất cả các ký tự trong dự đoán của mô hình không khớp với tất cả các ký tự trong văn bản tham chiếu chính hằng.

2.6.3 ROUGE Score

ROUGE [134], hay Recall-Oriented Understudy for Gisting Evaluation, là một tập hợp các số liệu và một gói phần mềm được sử dụng trong NLP để đánh giá phần mềm tóm tắt tự động và dịch máy.

Chuỗi con chung dài nhất (LCS) giữa đầu ra của mô hình và tham chiếu được đo bằng ROUGE-L. Thống kê dựa trên Chuỗi con chung dài nhất (LCS). ROUGE-L được xác định trong Công thức 2.9 là vấn đề chuỗi con chung dài nhất tự nhiên xem xét các điểm tương đồng về cấu trúc ở cấp độ câu và tự động xác định các n-gram đồng diễn dài nhất trong chuỗi. Tính toán Rouge-L được thể hiện trong Hình 2.16, trong đó đầu tiên tính độ chính xác sau đó là độ thu hồi và cuối cùng là tính toán ROUGE-L.

<u>Example 1</u>	<u>Example 2</u>
<ul style="list-style-type: none"> • <u>Reference sentence</u> <pre>def do_filter(parser , token) :</pre>	<ul style="list-style-type: none"> • <u>Reference sentence</u> <pre>def do_filter(parser , token) :</pre>
<ul style="list-style-type: none"> • <u>Predicted sentence</u> <pre>def do_filter(parser , token) :</pre>	<ul style="list-style-type: none"> • <u>Predicted sentence</u> <pre>def do_filter(parser , character) :</pre>
<ul style="list-style-type: none"> • Exact Match Accuracy: 1 	<ul style="list-style-type: none"> • Exact Match Accuracy: 0

Figure 2.15: Two examples for measuring the Exact Match Accuracy of two sentences

$$Rouge(L) = \frac{LCS(gram_n)}{count(gram_n)} \quad (2.9)$$

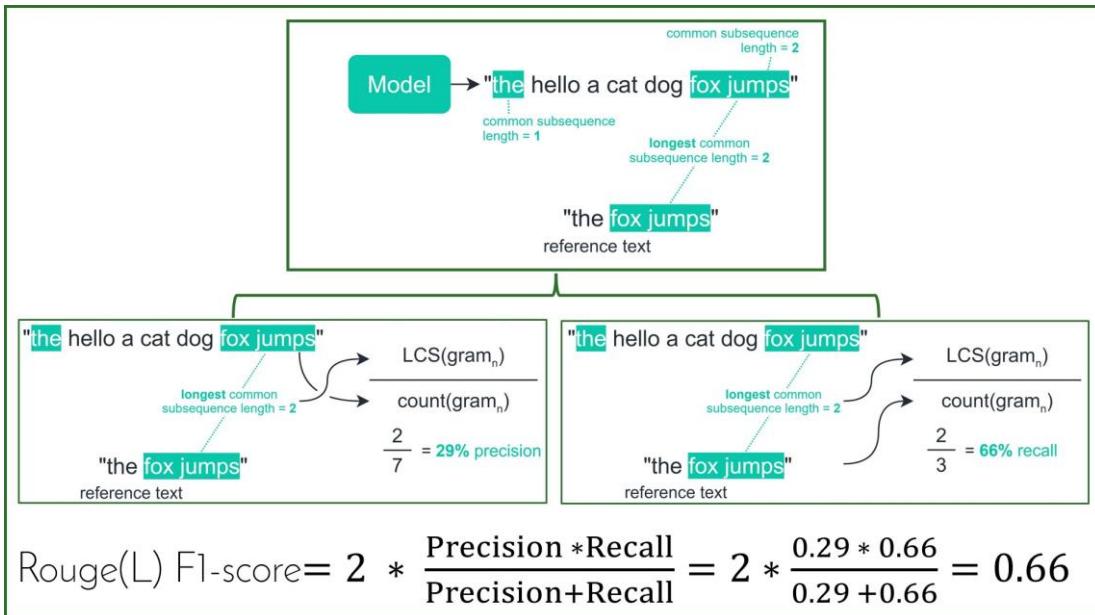


Figure 2.16: Example for calculating ROUGE-L

2.7 Error Correction and Code Enhancement

ROUGE [134], hay Recall-Oriented Understudy for Gisting Evaluation, là một tập hợp các số liệu và một gói phần mềm được sử dụng trong NLP để đánh giá phần mềm tóm tắt tự động và dịch máy.

Chuỗi con chung dài nhất (LCS) giữa đầu ra của mô hình và tham chiếu được đo bằng ROUGE-L. Thống kê dựa trên Chuỗi con chung dài nhất (LCS). ROUGE-L được xác định trong Công thức 2.9 là vấn đề chuỗi con chung dài nhất tự nhiên xem xét các điểm tương đồng về cấu trúc ở cấp độ câu và tự động xác định các n-gram đồng diễn dài nhất trong chuỗi. Tính toán Rouge-L được thể hiện trong Hình 2.16, trong đó đầu tiên tính độ chính xác sau đó là độ thu hồi và cuối cùng là tính toán ROUGE-L

2.7.1 Linting and Error Analysis

Linting là một dạng phân tích mã tĩnh tự động được thực hiện để phát hiện các lỗi về phong cách và lập trình trong mã. Linters phân tích mã để phát hiện nhiều loại lint khác nhau, bao gồm phân tích kiểu mã, linting bảo mật và phát hiện lỗi. Linting khác với định dạng vì linting phân tích cách mã chạy và phát hiện lỗi, trong khi định dạng chỉ tái cấu trúc cách mã xuất hiện. Linter Python được thiết kế riêng để phân tích mã Python và thực thi các biện pháp thực hành tốt nhất và xác định các vấn đề cụ thể của Python, chẳng hạn như thuật lề không đúng hoặc sử dụng các tính năng đã lỗi thời[138].

Có nhiều công cụ có sẵn để thực hiện linting và phân tích lỗi trong Python. Một số linter Python phổ biến nhất bao gồm PyLint, Flake8 và Pyflakes. PyLint là một linter có thể cấu hình cao, thực thi các tiêu chuẩn mã hóa và phát hiện lỗi. Flake8 và các công cụ và plugin khác, bao gồm Pyflakes, pycodestyle và McCabe. Pyflakes là các linter đơn giản và nhanh chóng, kiểm tra lỗi trong mã Python [139][140].

Sử dụng trình kiểm tra lỗi Python có thể giúp các nhà phát triển viết mã tốt hơn, khuyến khích áp dụng cách mã hóa nhất quán trong toàn bộ dự án và đơn giản hóa quy trình phát triển. Bằng cách hiểu sâu hơn về các lợi thế, tùy chọn được sử dụng rộng rãi và các kỹ thuật hiệu quả để kết hợp trình kiểm tra lỗi vào quy trình lập trình của bạn, bạn có thể cải thiện đáng kể khả năng đọc, khả năng bảo trì và chất lượng tổng thể của mã [141].

Flake8 1 là một công cụ Python kiểm tra kiểu và chất lượng của mã Python bằng cách kết hợp một số công cụ và plugin khác. Theo mặc định, Flake8 bao gồm các kiểm tra lỗi do Pyflakes, pycodestyle và McCabe cung cấp. Các plugin của bên thứ ba khác cũng có thể được thêm vào Flake8 để mở rộng chức năng của nó.

Sử dụng Flake8 giúp đảm bảo kiểu mã hóa nhất quán trên toàn bộ cơ sở mã của bạn, giúp bảo trì và hiểu mã dễ dàng hơn. Nó cũng có thể giúp phát hiện

các lỗi và lỗi tiềm ẩn sớm trước khi chúng gây ra sự cố trong quá trình sản xuất. Ngoài ra, Flake8 có thể được tích hợp vào quy trình phát triển của bạn, cung cấp các kiểm tra tự động về các thay đổi mã khi bạn làm việc trên chúng.

Có một số công cụ có sẵn trong Python để kiểm tra kiểu và chất lượng của mã. Như đã giới thiệu, một trong những công cụ này là Flake8, tương tự như Pylama. Pylama, lần lượt, bao gồm một số công cụ khác, chẳng hạn như Pyflakes, pycodestyle, McCabe và Radon, cung cấp đầu ra chung từ tất cả các plugin của nó. Cả Flake8 và Pylama đều dễ sử dụng và hiệu quả trong việc phân tích mã Python. Một công cụ khác đáng đề cập là Pylint 2, không chỉ kiểm tra kiểu mà còn phát hiện các tiêu chuẩn mã hóa, lỗi và các vấn đề khác trong mã Python.

Các công cụ này cùng nhau cung cấp cho các nhà phát triển nhiều tùy chọn để đảm bảo mã của họ đáp ứng các tiêu chuẩn về chất lượng và kiểu mong muốn. Bảng 2.2 hiển thị các trình kiểm tra lỗi Python nổi tiếng và xem xét những đóng góp của chúng trong việc nâng cao chất lượng mã Python. Các trình kiểm tra lỗi được công nhận rộng rãi này được phân loại như sau.

Vì vậy, cần phải sửa đổi để làm cho mã phù hợp vì mã Python hợp lệ đã được triển khai. Để sửa cú pháp mã đã tạo và tự động hóa việc sửa lỗi, có nhiều công cụ như yapf từ Google định dạng lại mã để tuân theo hướng dẫn về kiểu được chỉ định trong PEP8 [139].

2.7.2 Code Enhancement and Reformatting

Cải tiến và định dạng lại mã là một phần không thể thiếu của vòng đời phát triển phần mềm, cho phép các ứng dụng phát triển và thích ứng với các yêu cầu thay đổi trong khi vẫn duy trì các tiêu chuẩn chất lượng cao [142]. Có một số công cụ để sửa mã bao gồm Pylint và Black có thể được sử dụng để kiểm tra lỗi cú pháp, tiêu chuẩn mã hóa và các phương pháp hay nhất trong lập trình Python, đảm bảo mã sạch và chính xác, mang lại kết quả chính xác và phù hợp.

¹<https://flake8.pycqa.org/en/latest/>

²<https://readthedocs.org/projects/pylint/>

Table 2.2: Python Linters and Code Analysis Tools

Category	Linter
Linter	Pylint - A highly-configurable linter for comprehensive error checking and customizable coding guidelines enforcement.
General	Flake8 - A popular linter combining Pyflakes, pycodestyle, and McCabe complexity analysis for quick quality assurance.
Typing	MyPy - A user-friendly type checker focusing on static type checking and gradual typing support.
Typing	Pyright - A fast, lightweight type checker and linter by Microsoft, offering real-time development feedback.
Imports ordering	isort - A Python utility for sorting imports automatically, ensuring a clean and organized import section in your code.
Documentation	pydocstyle - Docstring style checker.
Security	Safety - checks Python dependencies for known security vulnerabilities and suggests the proper remediations for vulnerabilities detected.
Security	Bandit - a tool designed to find common security issues in Python code.
Code complexity	Xenon - a monitoring tool based on Radon. It monitors your code's complexity. Ideally, Xenon is run every time you commit code.
Code complexity	Radon - a tool that computes various metrics from the Python source code.

Ngoài ra, các công cụ phân tích tĩnh như pre-commit và analysis-tools-dev có thể giúp phát hiện lỗi và thực thi các tiêu chuẩn mã hóa trên các ngôn ngữ lập trình, tệp cấu hình và công cụ xây dựng. Ngoài ra, còn có các công cụ như Autopep8 giúp sửa các sự cố liên quan đến thuật lè, độ dài dòng và khoảng trắng. Các công cụ khác để sửa lỗi như thêm dấu phẩy cuối và cuối cùng là các công cụ để sắp xếp và định dạng các mục nhập trong mã Python[138].

Chapter 3

Literature Review

Trong những năm gần đây, các kỹ thuật học máy đã trở nên phổ biến trong các tác vụ tạo mã [143]. Các mô hình mạng nơ-ron, đặc biệt là các mô hình chuỗi-sang-chuỗi (Seq2Seq), đã cho thấy kết quả khả quan trong việc tạo mã từ các mô tả ngôn ngữ tự nhiên, mã giả hoặc các biểu diễn mã khác. Các cơ chế chú ý và kiến trúc biến đổi cũng đã được sử dụng để cải thiện nhận thức ngữ cảnh và hiệu suất của các mô hình này. Tuy nhiên, những thách thức như xử lý mã có độ dài thay đổi và tạo mã đúng về mặt cú pháp và ngữ nghĩa vẫn còn tồn tại.

Quá trình cho phép máy xác định biểu diễn ý nghĩa của nó được gọi là phân tích cú pháp ngữ nghĩa [5] [6] [7]. Công việc của phân tích cú pháp ngữ nghĩa là dịch các phát ngôn ngôn ngữ tự nhiên thành các biểu diễn dạng logic. Đó là ý nghĩa chính xác của một bài phát biểu, cũng như các cách tạo chương trình từ một tập hợp các hiện vật thiết lập các tiêu chí ngữ nghĩa và cú pháp cho mã đã tạo. Dịch máy, trả lời câu hỏi, quy nạp bản thể, lý luận tự động và tạo mã đều là các ví dụ về các ứng dụng phân tích cú pháp ngữ nghĩa.

Phân tích ngữ nghĩa, liên quan đến việc chuyển đổi các phát ngôn ngôn ngữ tự nhiên thành các dạng logic, là một thành phần chính trong việc hiểu ngôn ngữ tự nhiên. Bộ phân tích ngữ nghĩa ánh xạ các phát ngôn ngôn ngữ tự nhiên thành các biểu diễn ngữ nghĩa, còn được gọi là các dạng logic hoặc biểu diễn ý nghĩa [144]. Các biểu diễn này được thực hiện trên một môi trường hoặc ngữ cảnh. Phân tích ngữ nghĩa đã thu hút sự chú ý do tính hữu ích của nó trong nhiều ứng dụng khác nhau như trả lời câu hỏi, trích xuất quan hệ và giao diện ngôn ngữ tự nhiên.

Tạo mã là một phần không thể thiếu của phân tích ngữ nghĩa, một lĩnh vực nhằm mục đích chuyển đổi các phát ngôn ngôn ngữ tự nhiên thành các dạng logic ngôn ngữ chính thức. Trong những năm gần đây, đã có những tiến bộ đáng kể trong các nghiên cứu dựa trên ngôn ngữ tự nhiên về tạo mã nguồn thông minh [145][146]. Các kỹ thuật tạo mã bao gồm nhiều phương pháp khác nhau, bao gồm các kỹ thuật dựa trên cây và dựa trên học sâu. Các kỹ thuật này đóng vai trò quan trọng trong việc chuyển đổi các mô tả ngôn ngữ tự nhiên thành mã nguồn được viết bằng các ngôn ngữ lập trình đa năng như Python[147][148].

Trong Tạo mã ngôn ngữ dành riêng cho miền (DSL), các ngôn ngữ dành

riêng cho miền được thiết kế riêng cho các miền ứng dụng cụ thể và việc tạo mã từ các thông số kỹ thuật DSL đã thu hút được sự chú ý. Các kỹ thuật tạo mã DSL liên quan đến việc xác định ngữ pháp ngôn ngữ, cây cú pháp trừu tượng và các quy tắc chuyển đổi để chuyển đổi mã DSL cấp cao thành mã thực thi trong các ngôn ngữ đích. DSL cung cấp khả năng biểu đạt và trừu tượng cao nhưng đòi hỏi nhiều nỗ lực thiết kế và bảo trì ban đầu.

Ngoài ra, trong quá trình tạo mã cho Kỹ thuật hướng mô hình, Kỹ thuật hướng mô hình (MDE) là một phương pháp phát triển phần mềm nhấn mạnh vào việc sử dụng các mô hình để tự động tạo mã. MDE dựa vào các ngôn ngữ mô hình, chẳng hạn như Ngôn ngữ mô hình thống nhất (UML) và các chuyển đổi mô hình để tạo mã thực thi. Mặc dù MDE tạo điều kiện cho quá trình phát triển nhanh chóng và tính nhất quán, nhưng thách thức nằm ở việc tạo ra các chuyển đổi mô hình chính xác và hiệu quả.

Mặt khác, Biên dịch mã sang mã liên quan đến việc chuyển đổi mã từ ngôn ngữ lập trình này sang ngôn ngữ lập trình khác. Nó đặc biệt hữu ích cho việc chuyển phần mềm trên các nền tảng và hiện đại hóa các hệ thống cũ. Việc tạo mã tái cấu trúc nhằm mục đích cải thiện cấu trúc và khả năng bảo trì của mã bằng cách tự động áp dụng các tái cấu trúc, chẳng hạn như đổi tên biến, trích xuất phương pháp hoặc tối ưu hóa vòng lặp. Các phương pháp này nâng cao khả năng bảo trì phần mềm nhưng đòi hỏi các phân tích và chuyển đổi phức tạp theo từng ngôn ngữ.

Một trong những khía cạnh cốt lõi của vấn đề tạo mã là Tối ưu hóa và Tạo mã theo hiệu suất. Các kỹ thuật tạo mã theo tối ưu hóa tập trung vào việc tự động tạo mã được tối ưu hóa cho các số liệu hiệu suất cụ thể, chẳng hạn như thời gian thực thi hoặc mức sử dụng bộ nhớ. Các kỹ thuật này sử dụng phân tích tĩnh, lập hồ sơ và học máy để suy ra các điểm nghẽn về hiệu suất và tạo mã được tối ưu hóa. Mặc dù các kỹ thuật như vậy có thể nâng cao đáng kể hiệu quả ứng dụng, nhưng chúng thường đi kèm với sự gia tăng độ phức tạp và chi phí tính toán.

Trong bài đánh giá tài liệu này, chúng tôi sẽ khám phá các chủ đề liên quan đến việc tạo mã và phân tích ngữ nghĩa. Chúng tôi sẽ xem xét các kỹ thuật được sử dụng trong việc tạo mã[149], bao gồm các phương pháp dựa trên cây và dựa trên học sâu. Ngoài ra, chúng tôi sẽ thảo luận về các tập dữ liệu có sẵn để tạo mã và xem xét các công trình trước đây trong lĩnh vực này. Bằng cách đi sâu vào các chủ đề này, chúng tôi muốn đạt được sự hiểu biết toàn diện về những tiến bộ và thách thức trong việc tạo mã và phân tích ngữ nghĩa[150][151][152][153].

3.1 Code Generation Techniques

Tạo mã là quá trình tự động tạo mã nguồn từ biểu diễn cấp cao của chương trình. Có một số khó khăn trong vấn đề này vì đầu ra có cấu trúc được xác định rõ ràng và miền, cấu trúc của đầu vào và đầu ra không giống nhau.

Có nhiều kỹ thuật khác nhau được sử dụng trong việc tạo mã, bao gồm các phương pháp dựa trên cây và dựa trên học sâu. Các kỹ thuật dựa trên phân tích cú pháp ngữ nghĩa được sử dụng rộng rãi hơn. Các kỹ thuật dựa trên cây liên quan đến việc sử dụng cây cú pháp để tạo mã, trong khi các kỹ thuật học sâu sử dụng mạng nơ-ron để học cách ánh xạ giữa các mô tả ngôn ngữ tự nhiên và mã nguồn.

3.1.1 Semantic Parsing Based Techniques

Phân tích ngữ nghĩa là một lĩnh vực xử lý ngôn ngữ tự nhiên liên quan đến việc chuyển đổi các phát ngôn ngôn ngữ tự nhiên thành các biểu diễn ý nghĩa có thể hiểu được bằng máy, được gọi là các dạng logic. Mục tiêu của phân tích ngữ nghĩa là trích xuất ý nghĩa chính xác từ các phát ngôn ngôn ngữ tự nhiên, cho phép máy hiểu và thực thi chúng. Phân tích ngữ nghĩa có nhiều ứng dụng, bao gồm dịch máy, trả lời câu hỏi, quy nạp bản thể, lý luận tự động và tạo mã [9].

Đây là một số hướng giải quyết vấn đề phân tích ngữ nghĩa nói chung, cũng như các cách tiếp cận để giải quyết vấn đề tạo mã. Nhiều phương pháp phân tích ngữ nghĩa khác nhau được giới thiệu như sau:

- Sao chép ngữ cảnh và biến:

Phương pháp Sao chép ngữ cảnh và biến là một kỹ thuật trong phân tích ngữ nghĩa thần kinh để tạo mã. Tiến bộ này liên quan đến việc kết hợp ngữ cảnh xung quanh vào các mô hình thần kinh để phân tích ngữ nghĩa nhằm cải thiện sự mơ hồ. Phương pháp này sử dụng ngữ cảnh chương trình để viết mã có liên quan. Phương pháp này liên quan đến hệ thống mã hóa-giải mã trong đó bộ mã hóa nhận các phát ngôn đầu vào và biểu diễn định danh môi trường. Các định danh này bao gồm các chi tiết về biến và phương pháp. Trong quá trình giải mã, sự chú ý trước tiên được áp dụng cho ngôn ngữ tự nhiên, sau đó là các yếu tố môi trường. Sự chú ý hai bước này giúp khớp các từ trong ngôn ngữ với các định danh môi trường. Ngoài ra, một cơ chế sao chép có giám sát được sử dụng, cho phép mô hình sao chép các mã thông báo môi trường trong đầu ra, ngay cả khi chúng không được nhìn thấy trong quá trình đào tạo hoặc đầu vào.

Năm 2016, Gu et al. [154] đã đề xuất một cơ chế có giám sát sử dụng trọng số chú ý để tạo điều kiện sao chép các mã thông báo môi trường vào đầu ra. Cơ chế này cho phép mô hình sao chép các yếu tố môi trường như tên biến, ngay cả khi chúng không được tìm thấy trong dữ liệu đào tạo hoặc có mặt rõ ràng trong câu nói đầu vào. Bằng cách tận dụng trọng số chú ý, mô hình có được khả năng kết hợp thông tin môi trường có liên quan vào mã được tạo, tăng cường tính linh hoạt và khả năng thích ứng của nó vượt ra ngoài các ràng buộc của dữ liệu đào tạo và biểu thức đầu vào.

Năm 2018, Iyer et al. [155] đã giới thiệu một kiến trúc sáng tạo được thiết kế để tăng cường việc tạo mã có liên quan theo ngữ cảnh bằng cách tận dụng ngữ cảnh theo chương trình. Kiến trúc sử dụng hệ thống mã hóa-giải mã, trong đó bộ mã hóa lấy cả lời nói đầu vào và các biểu diễn đặc trưng của các định danh môi trường, bao gồm tên và kiểu biến và phương

thúc.

Quá trình giải mã tuân theo cơ chế chú ý hai bước: ban đầu tập trung vào đầu vào ngôn ngữ tự nhiên và sau đó là các biến và phương pháp của môi trường. Quá trình chú ý hai bước này hỗ trợ mô hình liên kết hiệu quả các từ trong ngôn ngữ tự nhiên với các biểu diễn tương ứng của các định danh môi trường, do đó tạo điều kiện thuận lợi cho việc tạo mã phù hợp hơn với ngữ cảnh lập trình xung quanh.

- Ngữ pháp trong Mô hình Phân tích Ngữ nghĩa Nơ-ron

Cấu trúc của ngôn ngữ có thể ảnh hưởng và hạn chế cách các mô hình mã hóa-giải mã nơ-ron giải mã thông tin trong quá trình phân tích ngữ nghĩa.

Năm 2016, Xiao và cộng sự đã đề xuất một kỹ thuật dựa trên trình tự để tạo các truy vấn cơ sở kiến thức [156]. Họ đã cấu trúc các dạng logic mục tiêu theo trình tự và chỉ ra lợi thế của việc bao gồm các ràng buộc ngữ pháp trong trình tự suy ra.

Năm 2017, Krishnamurthy và cộng sự đã chứng minh tiện ích của việc thực thi các ràng buộc kiểu trong quá trình tạo truy vấn cho dữ liệu bảng bán cấu trúc [157]. Họ nhấn mạnh rằng việc kết hợp liên kết thực thể rõ ràng trong quá trình mã hóa sẽ tăng cường độ chính xác bằng cách đảm bảo tạo ra dạng logic được gõ tốt.

Năm 2017, Yin và Neubig [6] và Ling và cộng sự [5] đã giới thiệu các phương pháp tạo mã ngôn ngữ lập trình cấp cao, như Python, đặc biệt khó khăn do biểu diễn động và ngữ nghĩa. Không giống như các nghiên cứu trước đây tập trung vào các ngôn ngữ dành riêng cho miền, các phương pháp này giải quyết các ngôn ngữ lập trình mục đích chung với độ phức tạp cao hơn. Ling và cộng sự

[5] đã trình bày một phương pháp tạo mã trình tự sang trình tự dựa trên dữ liệu sử dụng nhiều bộ dự đoán, bao gồm mô hình tạo và mô hình con trỏ để sao chép từ khóa từ đầu vào.

Dựa trên công trình của Ling và cộng sự [5], Yin và Neubig [6] nhấn mạnh rằng các ngôn ngữ lập trình mục đích chung phức tạp hơn về cấu trúc và cú pháp so với các trường hợp dành riêng cho miền. Họ đề xuất sử dụng cây cú pháp trừu tượng (AST) để hạn chế không gian tìm kiếm bằng cách tạo AST hợp lệ thông qua các chuỗi hành động bắt nguồn từ cú pháp cơ bản của ngôn ngữ. Mặc dù phương pháp này cải thiện việc tạo mã Python, nhưng họ thừa nhận những thách thức về khả năng mở rộng với các AST ngày càng phức tạp.

Năm 2019, Iyer và cộng sự [158] đã giới thiệu một phương pháp sử dụng giải mã dựa trên thành ngữ để giảm bớt các yêu cầu về dữ liệu đào tạo trong các hệ thống phân tích ngữ nghĩa bị hạn chế về ngữ pháp. Họ đã hợp lý hóa ngữ pháp bằng cách sử dụng các thành ngữ thường gặp nhất từ bộ đào tạo, hỗ trợ giám sát việc học bộ phân tích ngữ nghĩa. Tuy nhiên, có một giới hạn về mức độ kiến thức thành ngữ cải thiện hiệu suất do khả năng quá khớp và khả năng khai quát hóa giảm.

Năm 2019, Shin và cộng sự [159] đã theo đuổi việc khai thác thành ngữ mã để hỗ trợ các mô hình trong quá trình suy luận cấp cao và cấp thấp đồng thời. Họ đã tích hợp các thành ngữ mã đã khai thác vào ngữ pháp dưới dạng các toán tử mới, cho phép mô hình phát ra các thành ngữ hoàn chỉnh dưới dạng các cấu trúc chương trình. Bằng cách kết hợp các thành ngữ này vào vốn từ vựng của bộ tổng hợp thần kinh, mô hình đã đạt được khả năng tạo ra các cấu trúc chương trình cấp cao và cấp thấp một cách có thể hoán đổi cho nhau, phản ánh hành vi của lập trình viên con người.

Quá trình giải mã tuân theo cơ chế chú ý hai bước: ban đầu tập trung vào đầu vào ngôn ngữ tự nhiên và sau đó là các biến và phương pháp của môi trường. Quá trình chú ý hai bước này hỗ trợ mô hình liên kết hiệu quả các từ trong ngôn ngữ tự nhiên với các biểu diễn tương ứng của các định danh môi trường, do đó tạo điều kiện thuận lợi cho việc tạo mã phù hợp hơn với ngữ cảnh lập trình xung quanh.

- Ngữ pháp trong Mô hình Phân tích Ngữ nghĩa Nơ-ron

Cấu trúc của ngôn ngữ có thể ảnh hưởng và hạn chế cách các mô hình mã hóa-giải mã nơ-ron giải mã thông tin trong quá trình phân tích ngữ nghĩa.

Năm 2016, Xiao và cộng sự đã đề xuất một kỹ thuật dựa trên trình tự để tạo các truy vấn cơ sở kiến thức [156]. Họ đã cấu trúc các dạng logic mục tiêu theo trình tự và chỉ ra lợi thế của việc bao gồm các ràng buộc ngữ pháp trong trình tự suy ra.

Năm 2017, Krishnamurthy và cộng sự đã chứng minh tiện ích của việc thực thi các ràng buộc kiểu trong quá trình tạo truy vấn cho dữ liệu bảng bán cấu trúc [157]. Họ nhấn mạnh rằng việc kết hợp liên kết thực thể rõ ràng trong quá trình mã hóa sẽ tăng cường độ chính xác bằng cách đảm bảo tạo ra dạng logic được gõ tốt.

Năm 2017, Yin và Neubig [6] và Ling và cộng sự [5] đã giới thiệu các phương pháp tạo mã ngôn ngữ lập trình cấp cao, như Python, đặc biệt khó khăn do biểu diễn động và ngữ nghĩa. Không giống như các nghiên cứu trước đây tập trung vào các ngôn ngữ dành riêng cho miền, các phương pháp này giải quyết các ngôn ngữ lập trình mục đích chung với độ phức tạp cao hơn. Ling và cộng sự

[5] đã trình bày một phương pháp tạo mã trình tự sang trình tự dựa trên dữ liệu sử dụng nhiều bộ dự đoán, bao gồm mô hình tạo và mô hình con trả để sao chép từ khóa từ đầu vào.

Dựa trên công trình của Ling và cộng sự [5], Yin và Neubig [6] nhấn mạnh rằng các ngôn ngữ lập trình mục đích chung phức tạp hơn về cấu trúc và cú pháp so với các trường hợp dành riêng cho miền. Họ đề xuất sử dụng cây cú pháp trừu tượng (AST) để hạn chế không gian tìm kiếm bằng cách tạo AST hợp lệ thông qua các chuỗi hành động bắt nguồn từ cú pháp cơ bản của ngôn ngữ. Mặc dù phương pháp này cải thiện việc tạo mã Python, nhưng họ thừa nhận những thách thức về khả năng mở rộng với các AST ngày càng phức tạp.

Năm 2019, Iyer và cộng sự [158] đã giới thiệu một phương pháp sử dụng giải mã dựa trên thành ngữ để giảm bớt các yêu cầu về dữ liệu đào tạo trong các hệ thống phân tích ngữ nghĩa bị hạn chế về ngữ pháp. Họ đã hợp lý hóa ngữ pháp bằng cách sử dụng các thành ngữ thường gặp nhất từ bộ đào tạo, hỗ trợ giám sát việc học bộ phân tích ngữ nghĩa. Tuy nhiên, có một giới hạn về mức độ kiến thức thành ngữ cải thiện hiệu suất do khả năng quá khóp và khả năng khai thác giảm.

Năm 2019, Shin và cộng sự [159] đã theo đuổi việc khai thác thành ngữ mã để hỗ trợ các mô hình trong quá trình suy luận cấp cao và cấp thấp đồng thời. Họ đã tích hợp các thành ngữ mã đã khai thác vào ngữ pháp dưới dạng các toán tử mới, cho phép mô hình phát ra các thành ngữ hoàn chỉnh

dưới dạng các cấu trúc chương trình. Bằng cách kết hợp các thành ngữ này vào vốn từ vựng của bộ tổng hợp thần kinh, mô hình đã đạt được khả năng tạo ra các cấu trúc chương trình cấp cao và cấp thấp một cách có thể hoán đổi cho nhau, phản ánh hành vi của lập trình viên con người.

- Lập trình từ mô tả bằng các ngôn ngữ miền cụ thể phong phú và một số ít ví dụ đầu vào/đầu ra. Các mô hình như vậy để lập trình từ mô tả có thể được tìm thấy trong In the Neural Program Search [170] và Language to Logical Form with Neural Attention [171].

Các phương pháp phân tích cú pháp ngữ nghĩa này tương tự như tổng hợp chương trình từ mô tả, trong đó không gian chương trình bị giới hạn ở một số dạng có thứ tự.

Nhìn chung, trình phân tích cú pháp ngữ nghĩa ánh xạ các phát ngôn ngôn ngữ tự nhiên thành các biểu diễn ngữ nghĩa, thường được gọi là các dạng logic hoặc biểu diễn ý nghĩa. Các biểu diễn này được thực hiện trên một môi trường hoặc ngữ cảnh, chẳng hạn như cơ sở dữ liệu quan hệ hoặc cơ sở kiến thức. Phân tích cú pháp ngữ nghĩa có thể được chia thành hai loại: phân tích cú pháp ngữ nghĩa nông và phân tích cú pháp ngữ nghĩa sâu. Phân tích cú pháp ngữ nghĩa nông liên quan đến việc xác định các thực thể trong một phát ngôn và gắn nhãn cho chúng bằng các vai trò mà chúng đóng, trong khi phân tích cú pháp ngữ nghĩa sâu liên quan đến việc dịch ngôn ngữ tự nhiên trực tiếp thành ngôn ngữ biểu diễn ý nghĩa [172].

3.1.2 Tree-based Techniques

Các kỹ thuật dựa trên cây được coi là một trong những dạng phân tích cú pháp ngữ nghĩa theo nhiệm vụ, dịch đầu vào ngôn ngữ tự nhiên thành biểu diễn thực thi chính thức của máy. Các kỹ thuật này có thể biểu diễn mã dưới dạng Cây cú pháp trừu tượng (AST) có thể được mô tả là biểu diễn cây cú pháp của mã mục tiêu hoặc phiên bản đã được dọn dẹp của cây phân tích cú pháp nắm bắt cấu trúc của biểu thức, các thành phần điều khiển của chương trình. Các kỹ thuật dựa trên cây đã được sử dụng để tạo mã trong nhiều thập kỷ và đã được chứng minh là hiệu quả trong việc tạo mã cho các miền cụ thể[173].

Mục tiêu của AST là mô tả cấu trúc ngữ nghĩa của các câu trong ngôn ngữ máy tính dưới dạng cây. Ngữ nghĩa có thể được nêu bằng ngữ pháp thuộc tính, nhưng hầu hết các phương pháp tiếp cận ngữ nghĩa đều dễ hiểu hơn đáng kể khi dựa trên biểu diễn rõ ràng hơn về các cụm từ của ngôn ngữ[174]. Có các mẫu chuẩn cho các thành phần khác nhau của định nghĩa ngôn ngữ lập trình khi mô phỏng mã dưới dạng AST. Ngoài ra, khi định nghĩa mã dưới dạng AST, bạn cần biết tập hợp các phạm trù hoặc miền cú pháp và một tập hợp các quy tắc để mô tả cách kết nối các phạm trù này với nhau.

Tạo mã và phân tích ngữ nghĩa cần chuyển đổi các đầu vào không có cấu trúc (hoặc có cấu trúc một phần) thành các đầu ra có thể thực thi được, được định dạng tốt. Vì vậy, các nhà nghiên cứu đã sử dụng các mô hình chuỗi thành cây để tạo mã, với cây biểu diễn AST của mã nguồn mục tiêu [33] [171] [6] [7] [175] [176] [159] [177] [178] [173] ; vì họ muốn cải thiện quy trình tạo các đoạn mã của AST.

Việc sử dụng các phương pháp dựa trên cây trong công việc này mang lại nhiều lợi thế. Ví dụ, họ giải quyết thách thức của việc tạo mã bằng cách chuyển đổi đầu vào Ngôn ngữ tự nhiên đã cho thành AST tương ứng, do đó tăng cường độ chính xác thông qua việc áp đặt một cấu trúc mạch lạc vào đầu ra mã kết quả.

Ngoài ra, các phương pháp tiếp cận dựa trên cây có thể được áp dụng cho nhiều loại dữ liệu khác nhau, bao gồm dữ liệu có phân phối không theo quy ước. Hơn nữa, các kỹ thuật này trực quan, giúp các mô hình dự đoán phức tạp dễ hiểu hơn. Cuối cùng, các phương pháp dựa trên cây yêu cầu xử lý dữ liệu trước tối thiểu do không có các phép biến đổi biến mờ rộng [179].

Ngược lại, có những hạn chế liên quan đến việc sử dụng các phương pháp này. Việc biểu diễn mã dưới dạng AST tỏ ra khó khăn do số lượng nút thường mờ rộng trong cây, có xu hướng vượt quá độ dài của mô tả Ngôn ngữ tự nhiên tương ứng. Do đó, các phương pháp tiếp cận theo hướng cây gặp khó khăn trong việc tạo mã chính xác một cách nhất quán dựa trên dữ liệu đào tạo ít phổ biến hơn cho ngữ cảnh Ngôn ngữ tự nhiên đã cho. Hơn nữa, quá trình tạo AST vốn có tính đồng bộ, khiến cấu trúc đầu ra lệch khỏi cấu trúc đầu vào [180].

Tóm lại, các kỹ thuật dựa trên cây dễ hiểu và triển khai. Các kỹ thuật này có thể được sử dụng để tạo mã cho các miền cụ thể và chúng có thể được sử dụng để tạo mã dễ đọc và bảo trì. Mặt khác, các kỹ thuật dựa trên cây bị hạn chế về khả năng tạo mã cho các miền phức tạp. Họ cần rất nhiều nỗ lực thủ công để tạo cây cú pháp.

Năm 2017, Yin và Neubig đã đề xuất một kỹ thuật tạo mã no-ron theo cú pháp [6] xây dựng một cây cú pháp trừu tượng bằng cách áp dụng dần các hành động từ một mô hình ngữ pháp. Họ đã thiết kế một mô hình ngữ pháp xác suất để tạo AST. Ngữ pháp trừu tượng Python có một tập hợp các quy tắc sản xuất và AST được tạo ra bằng cách kết hợp nhiều quy tắc sản xuất, mỗi quy tắc bao gồm một nút đầu và nhiều nút con.

Năm 2017, Rabinovich và cộng sự đã đề xuất Abstract Syntax Networks [7] và các mạng này đã được giới thiệu cho cả nhiệm vụ tạo mã và phân tích cú pháp ngữ nghĩa. Ba tập dữ liệu phân tích cú pháp ngữ nghĩa đã được sử dụng: JOBS, GEO và ATIS. Tất cả các tập dữ liệu này đều bao gồm các câu hỏi ngôn ngữ tự nhiên và biểu diễn logic của các ký hiệu của chúng. Các đầu ra được biểu diễn dưới dạng AST, được xây dựng bởi một bộ giải mã có cấu trúc môđun được xác định động song song với cấu trúc của cây đầu ra.

Năm 2018, Yin và Neubig đã đề xuất TRANX [175] phân tích cú pháp phát ngôn thành biểu diễn ý nghĩa chính thức. TRANX được xây dựng thông qua hệ thống chuyển đổi và sử dụng hệ thống chuyển đổi này để chuyển đổi phát ngôn Ngôn ngữ tự nhiên thành Cây cú pháp trừu tượng (AST) thông qua một loạt các hành động xây dựng cây khi có đầu vào là phát ngôn Ngôn ngữ tự nhiên. Sau đó, bộ phân tích cú pháp được sử dụng để biến AST trung gian thành biểu diễn ý nghĩa dành riêng cho miền, kết thúc quá trình phân tích cú pháp. TRANX chấm điểm từng giả thuyết AST bằng mô hình xác suất do mạng no-ron chỉ định. Nhưng bộ phân tích cú pháp ngữ nghĩa no-ron, TRANX chỉ ra một vấn đề rõ ràng về sự không mạch lạc trong quá trình tạo và nhận kết quả

với tập dữ liệu CoNaLa là 24,30 đối với số liệu điểm BLEU.

3.1.3 Deep Learning based Techniques

Quá trình tạo mã nguồn được phân loại là văn bản sang văn bản hoặc trình tự sang trình tự và có thể đạt được thông qua việc sử dụng các mô hình Học sâu cho cả phát triển và bảo trì. Việc ứng dụng trí tuệ máy móc có khả năng hiểu và xây dựng các cấu trúc phần mềm phức tạp có tiềm năng đáng kể trong lĩnh vực Kỹ thuật phần mềm. Có nhiều mô hình trình tự sang trình tự khác nhau, có khả năng chuyển đổi mã dự định thành các trình tự khác nhau trong các miền thay thế [181]. Nghệ thuật lập trình nằm ở cốt lõi của phát triển phần mềm, thúc đẩy đổi mới và định hình bối cảnh kỹ thuật số của chúng ta. Tuy nhiên, việc tạo ra mã hiệu quả và đáng tin cậy vẫn là một nhiệm vụ tốn nhiều công sức và dễ xảy ra lỗi đối với các nhà phát triển.

Để giảm bớt những thách thức liên quan đến việc tạo mã thủ công, các nhà nghiên cứu và kỹ sư đã chuyển sự chú ý của họ sang các kỹ thuật học sâu như một giải pháp đầy hứa hẹn. Học sâu, một tập hợp con của trí tuệ nhân tạo, đã chứng minh được thành công đáng kể trên nhiều lĩnh vực, từ xử lý ngôn ngữ tự nhiên đến nhận dạng hình ảnh. Tiềm năng của nó trong việc tự động tạo mã đã thu hút được sự quan tâm đáng kể, mang đến triển vọng hấp dẫn là hợp lý hóa quá trình phát triển phần mềm và tăng năng suất. Sử dụng học sâu để giải quyết và xử lý nhiều vấn đề đã trở thành công nghệ quan trọng trong nhiều lĩnh vực, do đó, nhiều dự án nghiên cứu tập trung vào công nghệ học sâu và các mô hình tiên đào tạo.

Ngoài ra, Học chuyển giao đã chứng minh kết quả tuyệt vời trong việc tạo ra các mô hình mới tùy thuộc vào một mô hình được đào tạo trước khác. Học chuyển giao là quá trình tinh chỉnh một mô hình đã được đào tạo để thực hiện một công việc để thực hiện một nhiệm vụ khác. Một mô hình được đào tạo trước có thể được định nghĩa là một mạng được lưu trữ đã được đào tạo trên một tập dữ liệu lớn, thường là trên một nhiệm vụ quy mô lớn. Do đó, các nhà nghiên cứu gần đây [182] [178] [183] [184] [185] [186] [187] trong vấn đề tạo mã đã tập trung vào việc tinh chỉnh và đào tạo mô hình được đào tạo trước để tạo ra một mô hình hướng nhiệm vụ mới. Tiềm năng tuyệt vời của việc sử dụng học chuyển giao để điều chỉnh mô hình được đào tạo trước cho một công việc cụ thể hơn nữa cung cấp các kết quả và phát hiện nhất quán cho nhiệm vụ tạo mã seq2seq.

Tóm lại, các kỹ thuật học sâu đã cho thấy triển vọng lớn trong việc tạo mã từ các mô tả ngôn ngữ tự nhiên. Các kỹ thuật này sử dụng mạng no-ron để học cách ánh xạ giữa các mô tả ngôn ngữ tự nhiên và mã nguồn. Các kỹ thuật học sâu có thể tạo mã cho các miền phức tạp. Ngoài ra, các phương pháp này đòi hỏi ít nỗ lực thủ công hơn các kỹ thuật dựa trên cây. Mặt khác, các kỹ thuật học sâu đòi hỏi lượng dữ liệu lớn để đào tạo các mạng no-ron.

Các mô hình học sâu đã vượt trội hơn AST về độ chính xác. Việc sử dụng cây phân tích cú pháp từ đầu vào Ngôn ngữ tự nhiên vẫn chưa được khám phá tương đối, khiến các nhà nghiên cứu chuyển trọng tâm sang các kỹ thuật học sâu [188].

Các kỹ thuật này loại bỏ sự cần thiết phải xây dựng cây để tạo mã.

Nhiều nhà nghiên cứu đã làm việc trong nhiệm vụ tạo mã bằng cách sử dụng nhiều tập dữ liệu khác nhau như CoNaLa, DJANGO, ATIS, CodeSearchNet và các tập dữ liệu khác.

Năm 2016, Dong và Lapata đã đề xuất một phương pháp học từ các mô tả ngôn ngữ tự nhiên và biểu diễn ý nghĩa [171]. Họ đã sử dụng Mạng no-ron hồi quy (RNN) với các đơn vị bộ nhớ dài hạn ngắn (LSTM) để mã hóa các cụm từ và giải mã các cấu trúc logic để xem xét nhiệm vụ phân tích ngữ nghĩa. Họ đã tạo ra một kỹ thuật dựa trên mô hình mã hóa-giải mã tăng cường sự chú ý và kỹ thuật này có thể chuyển đổi các phát ngôn đầu vào thành các biểu diễn vectơ và tạo ra các dạng logic của chúng.

Điều này được thực hiện bằng cách điều kiện hóa các chuỗi hoặc cây đầu ra trên các biểu diễn vectơ. Các phát ngôn đầu vào được mã hóa và giải mã này cùng với các cấu trúc logic của chúng và lớp chú ý được sử dụng để kiểm soát trực tiếp quá trình tổng hợp chương trình. Kết quả thử nghiệm của họ cho thấy việc thêm bộ giải mã cây phân cấp và cơ chế chú ý vào mô hình đã nâng cao hiệu suất trên toàn diện.

Năm 2019, Yin và Neubig đã đề xuất mô hình Xếp hạng lại [176]. Họ đã sử dụng trình phân tích ngữ nghĩa TRANX trước đó để có được biểu diễn ý nghĩa của ngôn ngữ tự nhiên đầu vào dưới dạng Cây cú pháp trừu tượng. Họ đã thêm phương pháp xếp hạng lại để đưa ra biểu diễn ý nghĩa phù hợp nhất. Mô hình xếp hạng lại được trình bày như một phương pháp lặp lại nhanh để tăng cường độ chính xác của việc phân tích cú pháp và xếp hạng lại danh sách n-tốt nhất của biểu diễn ý nghĩa. Điều này có thể được thực hiện bằng cách sử dụng các đặc điểm được thiết kế để giải quyết các vấn đề trong các mô hình cơ sở. Mô hình này được sử dụng và có kết quả với bốn tập dữ liệu GEO, ATIS, DJANGO và CoNaLa. Kết quả thu được là 30,11 điểm BLEU khi phát triển và thử nghiệm với tập dữ liệu CoNaLa.

Năm 2019, Shin và cộng sự đã giới thiệu PATOIS [159], đây là một trình tổng hợp chương trình và cũng là một trình tổng hợp chương trình thần kinh đào tạo trình tổng hợp thần kinh dựa trên cây để sử dụng các thành ngữ mã trong khi tạo mã. Hệ thống PATOIS được xây dựng trên các mô hình tạo cấu trúc như mạng no-ron đồ thị và mô hình chuỗi thành cây.

Năm 2020, Sun và cộng sự đã đề xuất TreeGen [177], đây là một kiến trúc thần kinh dựa trên cây. TreeGen giải quyết vấn đề phụ thuộc lâu dài bằng cách lấy cơ chế chú ý trong Transformers và giới thiệu một trình đọc AST (en-coder) duy nhất để đưa các quy tắc ngữ pháp và cấu trúc AST vào mạng. TreeGen đã được thử nghiệm với chuẩn Python, HearthStone, cũng như hai chuẩn phân tích cú pháp ngữ nghĩa, ATIS và GEO. Mô hình này ghi lại độ chính xác 89,1 trong ATIS và 89,6 trong GEO.

Năm 2020, Xu và cộng sự đã đề xuất một mô hình học sâu bằng cách lấy mẫu lại dữ liệu, tinh chỉnh mô hình được đào tạo trước và sử dụng kết hợp kiến thức bên ngoài [178] để dự đoán mã python có thể thực thi. Để đưa kiến thức bên ngoài vào các

mô hình tạo mã, họ đã đề xuất một chiến lược không phụ thuộc vào mô hình dựa trên việc tăng cường dữ liệu, truy xuất và lấy mẫu lại dữ liệu, chiến lược này đã thu được kết quả mới về tác vụ tạo mã miền mở CoNaLa. Họ đã sử dụng tập dữ liệu CoNaLa-Mined [189], được khai thác tự động từ StackOverflow và chứa 600.000 cặp mã NL trong Python.

Họ đã sắp xếp tất cả các cặp theo điểm tin cậy và phát hiện ra rằng 100.000 ví dụ hàng đầu có mức độ chính xác của mã và tương quan mã NL tốt. Do đó, 100.000 cặp hàng đầu được chọn cho các bài kiểm tra. Họ đã tạo ra khoảng 13.000 cặp mã NL khác nhau (không lấy mẫu lại) từ tài liệu API Python sau khi xử lý trước. Họ cũng lấy mẫu cùng số cặp cho cài đặt lấy mẫu lại để cung cấp một so sánh công bằng. Họ sử dụng mô hình tạo mã NL-to-code TRANX [175] làm mô hình cơ bản, với xếp hạng lại giả thuyết [176]. Họ cũng sử dụng chuẩn hóa độ dài [190] để đảm bảo rằng tìm kiếm chùm tia không ưu tiên kết quả ngắn hơn kết quả dài hơn. Họ nhận được 30,69 điểm BLEU với kiến thức bên ngoài với mô hình API và khi họ thêm xếp hạng lại vào kiến thức bên ngoài với API, họ nhận được

32,26 số liệu điểm BLEU.

Năm 2021, Dahal và cộng sự đã đề xuất một bài báo [173] mô tả phân tích kiến trúc có cấu trúc cây và tác động của chúng đến vấn đề tạo mã. Họ đã chạy và thử nghiệm các mô hình văn bản-to-cây, cây-to-cây có cấu trúc và cây-to-cây tuyến tính trên các cây phân tích cú pháp dựa trên thành phần, trong đó mục tiêu của họ là tạo ra các AST tương ứng của mã. Họ đã sử dụng các tập dữ liệu CoNaLa và ATIS. Cây thành phần hoặc cây phụ thuộc đang mô tả cấu trúc cú pháp của đầu vào và những cây này có thể được sử dụng để thực hiện cẩn chỉnh cây con với mã đích khớp với AST và mang lại lợi ích cho công việc hạ lưu. Mô hình Cây sang Cây của họ đã đạt được kết quả tốt.

Năm 2021, Shin và cộng sự đã chứng minh rằng giải mã bị ràng buộc của các mô hình ngôn ngữ phức tạp có thể cho phép diễn đạt lại các phát ngôn của người dùng thành một ngôn ngữ con được quy định, sau đó có thể được ánh xạ thành một biểu diễn cụ thể cho tác vụ [33]. Trong bài báo của mình, họ đã chỉ ra rằng các mô hình ngôn ngữ bị ràng buộc có thể tạo ra và hỗ trợ cho một số ít trình phân tích cú pháp ngữ nghĩa.

Năm 2021, Orlanski và Gittens đã làm việc để mở rộng tập dữ liệu CoNaLa ban đầu để bao gồm các nội dung câu hỏi văn bản đa phương thức và do đó bao gồm thông tin ngữ cảnh có liên quan mà chúng chứa như đâu vào, đâu ra và các thư viện bắt buộc [183]. Họ đã phát triển một mô hình biến đổi mới được đào tạo trong mô hình mã hóa giải mã BART [191]. Đối với cả các trường hợp được chú thích và khai thác trong ngữ liệu CoNaLa, họ đã lấy được các nội dung văn bản này từ Stackoverflow. Sau đó, họ sử dụng các thân câu hỏi và các ý định nối tiếp làm đầu vào cho một mô hình ngôn ngữ được đào tạo trước khổng lồ, sau đó sử dụng tìm kiếm chùm để xây dựng đoạn mã trả lời. Họ sử dụng Python và gói transformer của HuggingFace để xây dựng mô hình của họ. Cuối cùng, để tạo văn bản, họ sử dụng mô hình BART với một lớp tuyến tính và một độ lệch riêng biệt. Họ đạt được

26,24 điểm BLEU khi sử dụng mô hình cơ sở BART và khi họ sử dụng mô hình BART với dữ liệu được khai thác, họ đạt được 30,55 điểm BLEU.

Năm 2021, Norouzi và cộng sự đã chỉ ra rằng các mô hình seq2seq dựa trên transformer có thể cạnh tranh hoặc vượt trội hơn các mô hình được tạo riêng cho mục đích tạo mã [182]. Họ đã tạo ra một mô hình transformer seq2seq và họ xây dựng mô hình này bằng cách tinh chỉnh mô hình transformer BERT được đào tạo trước như một bộ mã hóa và bộ giải mã là bộ giải mã transformer ban đầu với bộ giải mã transformer 4 lớp. Chìa khóa là tạo ra một mô hình mới và kết hợp các tập hợp ngũ liệu đơn ngũ tương đối lớn của biểu diễn ý nghĩa với các bộ mã hóa được đào tạo trước quy mô lớn truyền thống.

Họ đạt điểm BLEU cao nhất với tập dữ liệu CoNaLa đạt tới 32,57. Một bộ chuyển đổi mô hình Seq2Seq với một chút chuyên biệt trước đó có khả năng đạt được kết quả vượt trội hoặc cạnh tranh với các mô hình được phát triển đặc biệt để tạo mã và phân tích ngũ nghĩa bằng cách tận dụng một ngũ liệu đơn ngũ đủ lớn của ngôn ngữ lập trình.

Năm 2022, Beau và Crabbé đã phát triển một mô hình tạo mã mới có kiến trúc mã hóa-giải mã [184]. Họ sử dụng BERT làm mã hóa và giải mã dựa trên ngũ pháp. Đây là một số thay đổi trong kiến trúc TranX [175] seq2seq để tạo mã từ mô tả ngôn ngữ tự nhiên. Kiến trúc đề xuất của họ có thể thu được AST bị hạn chế bởi ngũ pháp ngôn ngữ lập trình. Họ đã đào tạo và thử nghiệm mô hình của mình trên các tập dữ liệu CoNaLa và DJANGO. Hệ thống chuyển đổi này mở ra để đảm bảo tạo ra mã cú pháp chính xác. Nghiên cứu của họ nhấn mạnh tầm quan trọng của các hạn chế về ngũ pháp cũng như các kỹ thuật cụ thể để quản lý biến, đặt tên danh sách và nhập. Họ đã đạt điểm

34,2 BLEU.

Khảo sát tài liệu này và công trình liên quan về tạo mã cho thấy nhiều kỹ thuật và phương pháp khác nhau được sử dụng để giải quyết nhiệm vụ đầy thách thức là tự động tạo mã thực thi từ các biểu diễn cấp cao. Các kỹ thuật tạo mã cổ điển, chẳng hạn như các phương pháp dựa trên mẫu và dựa trên trình phân tích cú pháp, đã đặt nền tảng cho lĩnh vực này nhưng có thể thiếu tính linh hoạt và khả năng thích ứng cần thiết cho các cấu trúc mã phức tạp.

Trong những năm gần đây, sự ra đời của các phương pháp dựa trên máy học, đặc biệt là các mô hình Seq2Seq nơ-ron với cơ chế chú ý, đã cho thấy kết quả đầy hứa hẹn trong việc tạo mã từ các mô tả ngôn ngữ tự nhiên và mã giả. Các phương pháp này có tiềm năng thu hẹp khoảng cách giữa các mô tả có thể đọc được bằng con người và mã có thể thực thi bằng máy, mặc dù chúng vẫn gặp phải những thách thức trong việc xử lý mã có độ dài thay đổi và đảm bảo tính chính xác về cú pháp và ngũ nghĩa.

Hơn nữa, nghiên cứu đã mở rộng để phục vụ cho các miền ứng dụng cụ thể, chẳng hạn như tạo mã ngôn ngữ dành riêng cho miền, kỹ thuật dựa trên mô hình, dịch mã sang mã và tạo mã dựa trên hiệu suất. Mỗi miền này đều có bộ yêu cầu và ràng buộc riêng, đòi hỏi các giải pháp phù hợp.

Tính khả dụng ngày càng tăng của các tập dữ liệu mã quy mô lớn cũng đã đóng góp đáng kể vào những tiến bộ trong việc tạo mã. Các nhà nghiên cứu đã sử dụng nhiều tập dữ liệu khác nhau, từ kho lưu trữ mã Python đến các chuẩn mực dịch thuật đa ngôn ngữ, để đào tạo và đánh giá các mô hình của họ. Đánh

giá của con người vẫn là một thành phần quan trọng trong việc đánh giá chất lượng và tính chính xác của mã được tạo ra, bổ sung cho các số liệu đánh giá tự động truyền thống.

Tóm lại, khảo sát tài liệu chứng minh sự phát triển và đổi mới liên tục trong các kỹ thuật tạo mã, được thúc đẩy bởi sự tích hợp của máy học, kiến thức chuyên ngành và các tập dữ liệu quy mô lớn. Tuy nhiên, vẫn còn những thách thức liên quan đến chất lượng mã, khả năng mở rộng và khả năng thích ứng với nhiều ngôn ngữ và mô hình lập trình.

Nghiên cứu trong tương lai trong lĩnh vực này nên tập trung vào việc phát triển các phương pháp tiếp cận kết hợp tận dụng thế mạnh của các kỹ thuật khác nhau, khám phá các phương pháp đánh giá mới và giải quyết các yêu cầu tạo mã trong thế giới thực để đạt được những bước tiến đáng kể hướng tới việc tạo mã tự động, đáng tin cậy và hiệu quả.

3.2 Code Generation Datasets

Trong luận án này, chúng tôi đã sử dụng các tập dữ liệu phổ biến và nổi tiếng, CoNaLa, Django và CodeSearchNet. Các tập dữ liệu này được tạo ra với mục đích tạo mã từ các mô tả ngôn ngữ tự nhiên tương ứng.

3.2.1 CoNaLa Dataset

Một trong những tập dữ liệu phổ biến nhất trong tác vụ tạo mã được gọi là CoNaLa

[189] và được tạo ra bởi Carnegie Mellon University NeuLab và STRUDEL Lab. Nó được gọi là CoNaLa theo tên của Code/Natural Language Challenge và có sẵn tại: <https://conala-corpus.github.io/>. Tập dữ liệu này có 2.879 cặp mã NL được chú thích với khoảng 600K cặp được khai thác từ hơn 40.000 câu hỏi stackoverflow riêng biệt trong tập dữ liệu. Nó có đầu vào là mô tả ngôn ngữ tự nhiên với đầu ra dự kiến là mã python tương ứng cho đầu vào cụ thể này như thể hiện trong Hình 3.1. Ngoài ra, bảng 3.1 cũng hiển thị một số ví dụ về cặp mã NL trong tập dữ liệu CoNaLa.

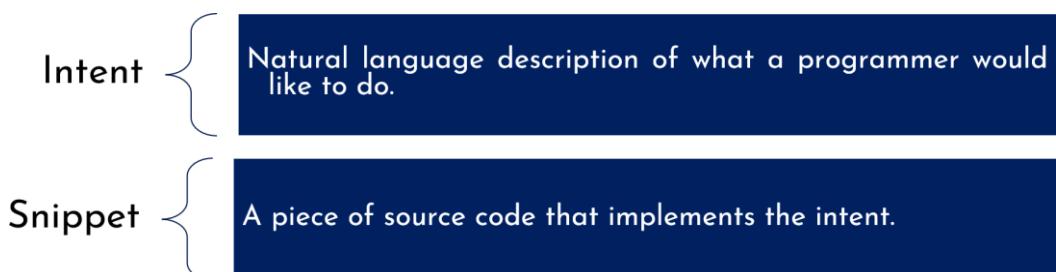


Figure 3.1: The CoNaLa dataset contents

Table 3.1: CoNaLa Dataset NL-Code pairing examples

Intent (Natural Language)	Snippet (Code)
convert a list to a dictionary in python	<code>b = dict(zip(a[0::2], a[1::2]))</code>
sort a list of nested lists	<code>l.sort(key=sumnested)</code>
how to get the size of a string in python?	<code>print(len('string'))</code>

Chúng tôi đã theo dõi các nhà nghiên cứu khác trong công việc của họ để trích xuất các ví dụ chính xác nhất từ 600K bản ghi, chỉ với mục đích và đoạn mã để làm việc trên việc tạo mã.

Trong luận án này, chúng tôi đã làm việc với hai tập hợp con từ tập dữ liệu khai thác CoNaLa:

1. CoNaLa

The first dataset contains 13K records of intent and snippet pairs. It is available at: <https://huggingface.co/datasets/AhmedSSoliman/CoNaLa>

2. CoNaLa-Large

This version of CoNaLa has 26K records of intent and snippet pairs. It is available at:

<https://huggingface.co/datasets/AhmedSSoliman/CoNaLa-Large>

3.2.2 DJANGO Dataset

Bộ dữ liệu DJANGO [192] là một trong những bộ dữ liệu được sử dụng phổ biến nhất trong tác vụ tạo mã. Nó có khoảng 19K ví dụ. Mỗi ví dụ dữ liệu được tạo thành từ một dòng mã Python và một mô tả ngôn ngữ tự nhiên có chú thích. Các ví dụ này được chia thành 16000 chú thích đào tạo, 1000 chú thích phát triển và 1805 chú thích thử nghiệm trong bộ dữ liệu Django.

Bảng 3.2 cung cấp nhiều trường hợp cặp mã NL DJANGO. Đầu vào là văn bản ngôn ngữ tự nhiên và đầu ra là mã tương ứng cho đầu vào này.

Table 3.2: Code generation examples in the DJANGO dataset

Natural Language	Code
define the function do_filter with 2 arguments: parser and token.	def do_filter (parser , token) :
convert priority into a floating point integer, substitute it for priority.	priority = float (priority)
define the method as_bytes with arguments self and unixfrom set to boolean False.	def as_bytes (self , unixfrom = False):

Bộ dữ liệu DJANGO là một nguồn tài nguyên có giá trị cho nghiên cứu xử lý ngôn ngữ tự nhiên (NLP) và học máy, đặc biệt là trong lĩnh vực các tác vụ liên quan đến mã. Đây là một bộ dữ liệu quy mô lớn bao gồm các cặp mô tả ngôn ngữ tự nhiên và các đoạn mã tương ứng được trích xuất từ các dự án nguồn mở được xây dựng bằng cách sử dụng khung web Django. Bộ dữ liệu được biên soạn để tạo điều kiện thuận lợi cho việc phát triển và đào tạo các mô hình NLP có khả năng hiểu được mối quan hệ giữa ngôn ngữ con người và mã lập trình. Với bộ sưu tập các ví dụ mã dành riêng cho Django đa dạng và phong phú, bộ dữ liệu DJANGO cho phép các nhà nghiên cứu khám phá và phát triển các kỹ

thuật trong tóm tắt mã, tạo mã và tìm kiếm mã, cuối cùng là thu hẹp khoảng cách giữa ngôn ngữ tự nhiên và ngôn ngữ lập trình để cải thiện quy trình phát triển phần mềm.

DJANGO dataset is available at:

<https://github.com/odashi/ase15-django-dataset>

Also, we uploaded DJANGO dataset on the huggingface hub to be available at:

<https://huggingface.co/datasets/AhmedSSoliman/DJANGO>

3.2.3 CodeSearchNet Dataset

CodeSearchNet [193] là sáng kiến nghiên cứu hợp tác do GitHub, OpenAI và các đối tác khác phát triển với mục đích nâng cao khả năng tìm kiếm và hiểu mã trong lĩnh vực phân tích và xử lý văn bản. Dự án nhằm mục đích hỗ trợ các nhà phát triển trong việc định vị các đoạn mã và tài liệu có liên quan, hợp lý hóa quy trình tìm kiếm các ví dụ mã hữu ích và các tài nguyên liên quan.

Mục tiêu chính của CodeSearchNet là thu hẹp khoảng cách giữa ngôn ngữ con người và ngôn ngữ lập trình, vì các công cụ tìm kiếm thông thường không được tối ưu hóa cho các truy vấn liên quan đến mã. Bằng cách khai thác các khả năng của NLP và máy học, CodeSearchNet đã tìm cách tạo ra các công cụ và mô hình hiệu quả hơn để hỗ trợ khám phá và tìm kiếm mã.

Để thực hiện được điều này, dự án đã xây dựng một tập dữ liệu mở rộng bao gồm các cặp truy vấn ngôn ngữ tự nhiên và các đoạn mã tương ứng của chúng. Tập dữ liệu này được tuyển chọn bằng cách trích xuất thông tin từ các kho lưu trữ công khai trên GitHub, tạo ra một bộ sưu tập đa dạng các ví dụ mã trên nhiều ngôn ngữ lập trình, thư viện và miền khác nhau.

CodeSearchNet đã trình bày nhiều ứng dụng tiềm năng và trường hợp sử dụng, bao gồm:

1. Công cụ tìm kiếm mã: Bộ dữ liệu cho phép các nhà nghiên cứu và nhà phát triển đào tạo các công cụ tìm kiếm mã có khả năng hiểu các truy vấn ngôn ngữ tự nhiên và truy xuất các đoạn mã có liên quan. Các công cụ tìm kiếm này tỏ ra vô cùng hữu ích trong việc hỗ trợ các nhà phát triển tìm mã giải quyết các vấn đề cụ thể hoặc đáp ứng các yêu cầu của dự án.

2. Tạo mã: Bằng cách sử dụng các mô hình học máy tiên tiến, CodeSearchNet có tiềm năng tạo các đoạn mã dựa trên mô tả ngôn ngữ tự nhiên, tự động hóa một số phần của quy trình mã hóa.

3. Tài liệu mã: Bộ dữ liệu có thể được tận dụng để nâng cao chất lượng tài liệu mã bằng cách đề xuất các ví dụ mã có liên quan cho các tác vụ lập trình cụ thể.

4. Hiểu mã: Các nhà nghiên cứu có thể sử dụng bộ dữ liệu để hiểu sâu hơn về mối quan hệ giữa ngôn ngữ tự nhiên và mã, thúc đẩy sự hiểu biết của chúng ta về cách con người thể hiện các khái niệm lập trình.

Ngoài tác động tức thời, CodeSearchNet còn khơi dậy sự quan tâm rộng rãi hơn đối với việc phát triển các mô hình liên quan đến lập trình và dẫn đến những tiến bộ đáng kể trong lĩnh vực Code-ML (Mã + Học máy). Bản chất hợp tác của dự án nhằm thúc đẩy sự đổi mới và hợp tác giữa các nhà nghiên cứu, nhà phát triển và cộng đồng nguồn mở rộng lớn hơn. Như thường lệ đối với các dự án nghiên cứu, tập dữ liệu và mô hình liên tục được cập nhật và cải tiến theo thời gian, góp phần vào tiến trình chung trong NLP, tìm kiếm mã và nghiên cứu liên quan đến mã.

Chapter 4

Proposed Approach

1.1 Introduction

Tạo mã là nhiệm vụ tự động tạo và xây dựng mã từ mô tả của con người. Nhiệm vụ này là một trong những nhiệm vụ thú vị có thể hỗ trợ các lập trình viên và nhà phát triển học ngôn ngữ lập trình và tương tác dễ dàng với các hệ thống mới khác.

Trong luận án này, các mô hình mới được đề xuất để giải quyết vấn đề tạo mã. Một trong những mô hình này sử dụng mô hình dịch máy để thực hiện học chuyển giao trên tập dữ liệu tạo mã. Các mô hình còn lại là mô hình mã hóa-giải mã như thể hiện trong Hình 4.1 sử dụng mô hình ngôn ngữ biến đổi được đào tạo trước làm bộ mã hóa và Bộ giải mã dịch máy thần kinh Marian làm bộ giải mã. Các thí nghiệm đã được thực hiện với nhiều mô hình được đào tạo trước khác nhau để xây dựng các mô hình lai mới.

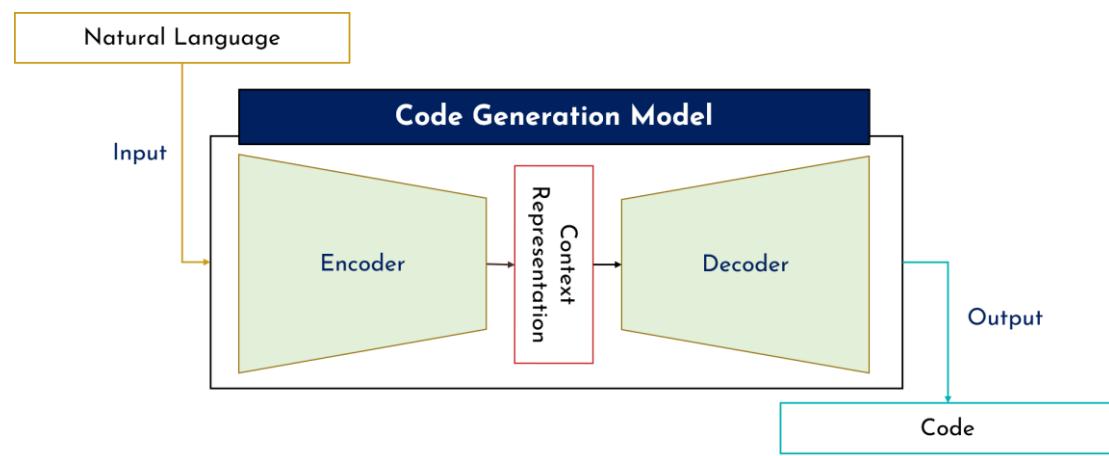


Figure 4.1: Code generation seq2seq model

1.2 MarianCG Model

Marian [194] là công cụ cốt lõi cho các dịch vụ Dịch máy thần kinh Microsoft Translator. Marian là một khuôn khổ Dịch máy thần kinh hiệu quả, mã nguồn mở miễn phí và độc lập, là một công cụ phân biệt tự động tích hợp dựa trên đồ thị tính toán động. Nó đang được sử dụng trong một số dự án và là công cụ dịch thuật và đào tạo chính cũng như được sử dụng bởi nhiều doanh nghiệp, tổ chức và nhóm nghiên cứu.

Marian đã được phòng thí nghiệm NLP của Đại học Helsinki sử dụng để đào tạo hàng trăm mô hình dịch thuật bằng cách sử dụng dữ liệu song song thu được từ Opus và các mô hình đó sau đó đã được mã nguồn mở và được gọi là MarianMT 1. Các mô hình MarianMT được tạo ra bởi bộ công cụ Marian, cho phép đào tạo và dịch thuật nhanh chóng. Mỗi thành phần của mô hình là một bộ mã hóa-giải mã biến áp có sáu lớp. Hiệu suất của mỗi mô hình được ghi lại trên một thẻ mô hình.

Dịch thuật là một trong số nhiều công việc có thể được thể hiện dưới dạng thử thách trình tự sang trình tự, một khuôn khổ mạnh mẽ cũng có thể được sử dụng cho các vấn đề về thị giác và âm thanh. Nhiều mô hình chuyển đổi dịch máy như Google T5 [105], Facebook BART [191] và MarianMT [194] có thể cung cấp kết quả rất đáng tin cậy.

Các mô hình chuyển đổi tiền đào tạo hiện nay được sử dụng rộng rãi cho nhiều tác vụ khác nhau và chúng đã được sử dụng trong việc tạo mã trong những năm qua và đạt được kết quả tốt. Chúng tôi đề xuất MarianCG, một mô hình tạo mã lấy cảm hứng từ mô hình chuyển đổi dịch máy như thể hiện trong Hình 4.2.

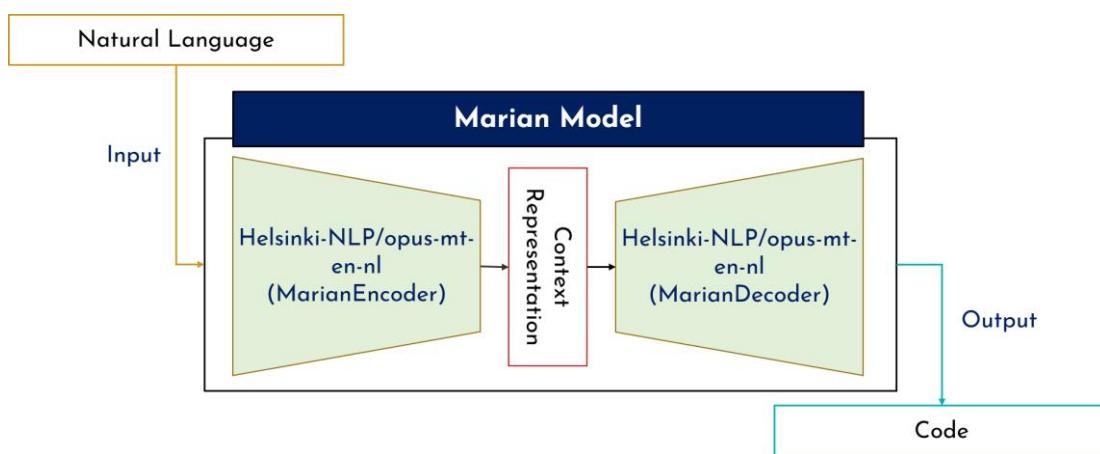


Figure 4.2: Marian Model

Việc triển khai của chúng tôi phụ thuộc vào các điểm kiểm tra được đào tạo trước của máy biến áp cho bộ mã hóa và Bộ giải mã Marian từ mô hình Dịch máy thần kinh Marian

[194]. Chúng tôi đã cố gắng chứng minh rằng mô hình dịch máy có thể được sử dụng và sử dụng như một mô hình tạo mã.

¹https://huggingface.co/docs/transformers/model_doc/marian

Vì vậy, chúng tôi đã sử dụng MarianMT vì các tính năng dịch thuật nâng cao của nó để giải quyết vấn đề tạo mã. Bên cạnh đó, chúng tôi đã xây dựng các mô hình lai sử dụng các mô hình ngôn ngữ được đào tạo trước [195] để tận dụng các mô hình này để tạo ra thế hệ mã mới theo một số cách.

Mô hình MarianCG được xây dựng và phát triển bằng cách sử dụng Marian Neural Machine Translation (MarianNMT). Chúng tôi đã tinh chỉnh bộ biến đổi MarianMT, một mô hình được đào tạo trước từ Helsinki-NLP và có được mô hình MarianCG của chúng tôi như thể hiện trong Hình 4.3. Đây là một bộ biến đổi chú ý nhiều đâu với phương pháp học không có lần bắn nào, quan sát các mẫu không được hiển thị trong quá trình đào tạo và dự đoán các câu là đâu ra đúng. Sau khi đào tạo và tinh chỉnh cùng một kiến trúc mô hình trên nhiều dữ liệu hơn, trở thành 26K cặp NL-Code và kích thước lô trỏ nên nhỏ hơn, điều này dẫn đến hiệu suất tốt hơn. So với quá trình đào tạo trước đây của chúng tôi về mô hình MarianCG, chúng tôi đã có được kết quả mới và được cải thiện, trong đó điểm BLEU trở thành 34,43 và điểm ROUGE.

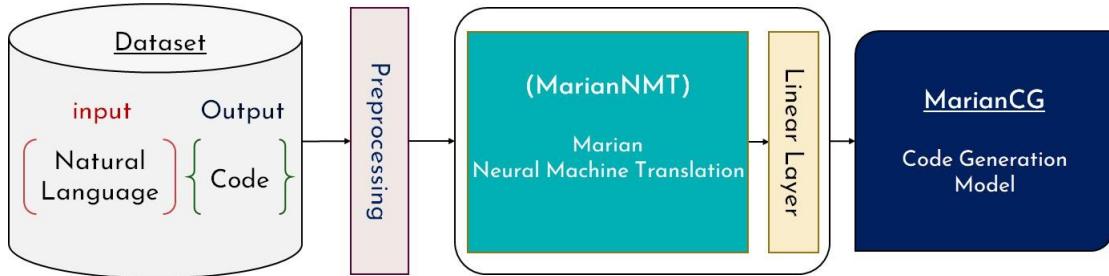


Figure 4.3: MarianCG model for Code Generation

MarianCG được đào tạo và tinh chỉnh trên mô hình MarianMT được xây dựng bằng MarianNMT. MarianNMT cho phép đào tạo và dịch nhanh. Hình 4.4 cho thấy kiến trúc của mô hình tạo mã, MarianCG. Nó bắt đầu bằng cách tải tập dữ liệu, sau đó là giai đoạn tiền xử lý của đầu vào và đầu ra như thể hiện trong Hình 4.5. Tiền xử lý bao gồm mã hóa câu và sau đó nhúng từng mã thông báo và nhúng vị trí cho từng mã thông báo để tìm hiểu từng vị trí của tất cả các mã thông báo liên quan đến mã thông báo cụ thể. Ngoài ra, việc đệm và cắt bớt là một phần của giai đoạn tiền xử lý. Cuối cùng, các câu đầu vào và đầu ra được chèn trực tiếp vào mô hình mã hóa-giải mã. Mô hình MarianCG bao gồm một ngăn xếp gồm 6 lớp trong bộ mã hóa và một ngăn xếp gồm 6 lớp trong bộ giải mã. Mô hình MarianCG tương tự như BartForConditionalGeneration với một vài sửa đổi nhỏ trong đó MarianCG có nhúng vị trí tĩnh (hình sin) và không nhúng chuẩn hóa lớp. Ngoài ra, token đầu tiên trong tác vụ tạo là pad token có 0 làm token nhúng. Sau attention softmax, trọng số attention của bộ mã hóa được sử dụng để tính toán trọng số trung bình trong các đầu tự chú ý.

MarianCG là mô hình PyTorch với mã hóa và triển khai bộ biến đổi dịch máy thần kinh Marian. Hình 4.6 cho thấy biểu diễn ví dụ thông qua mô hình MarianCG.

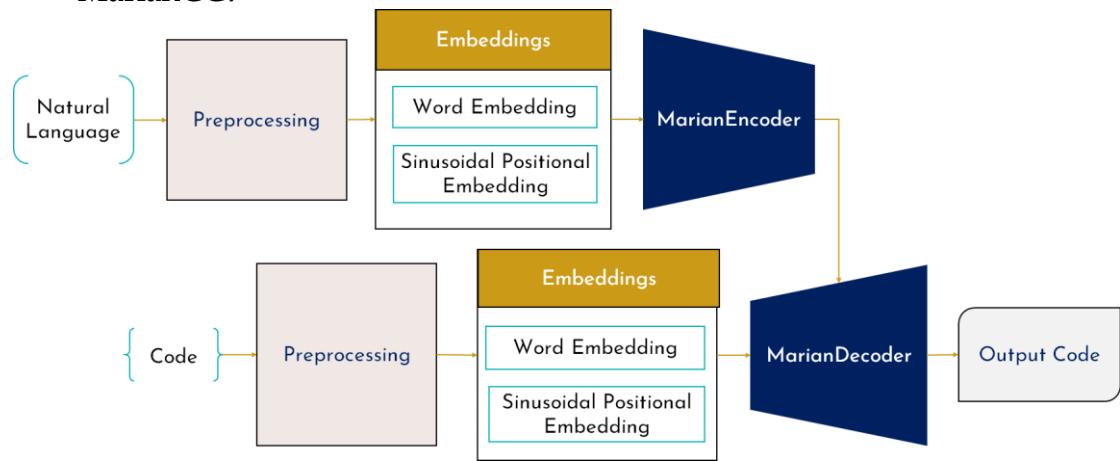


Figure 4.4: MarianCG Model Architecture

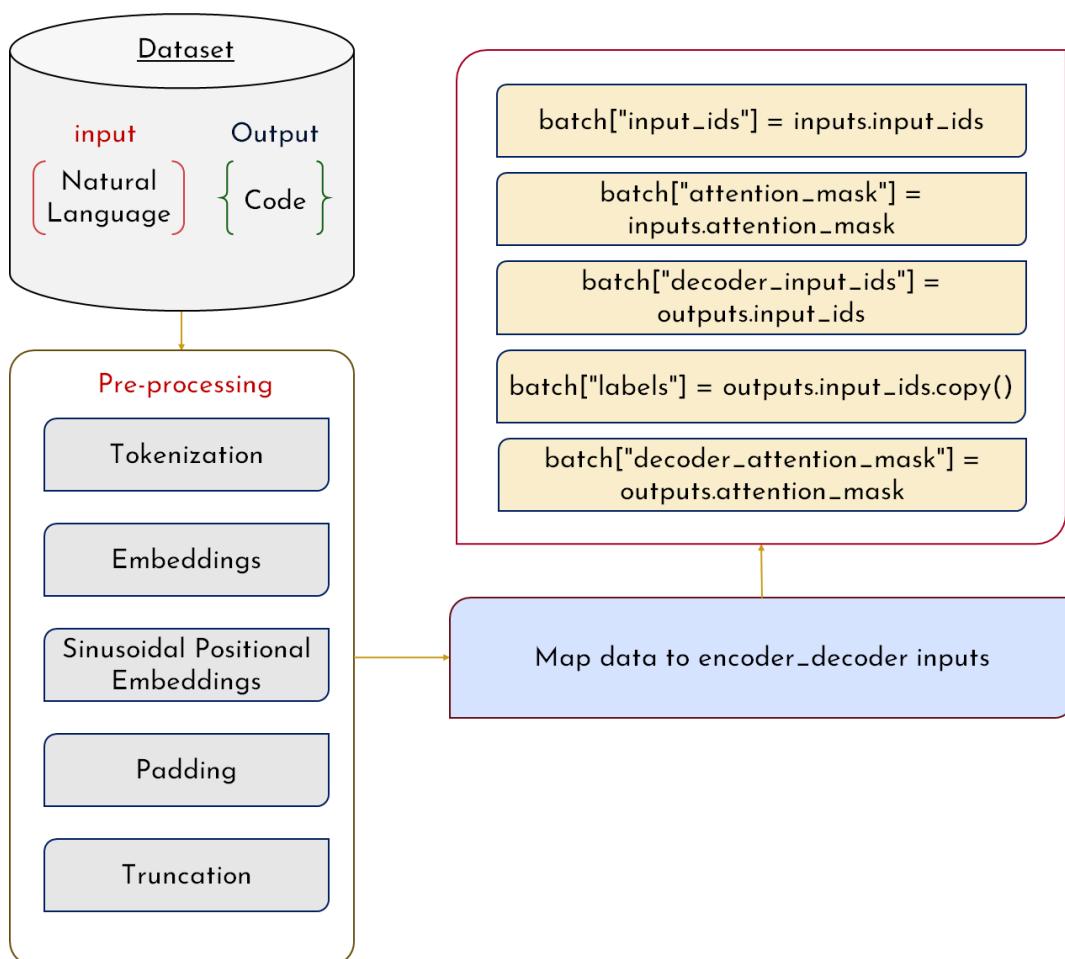


Figure 4.5: Preprocessing phase for the data

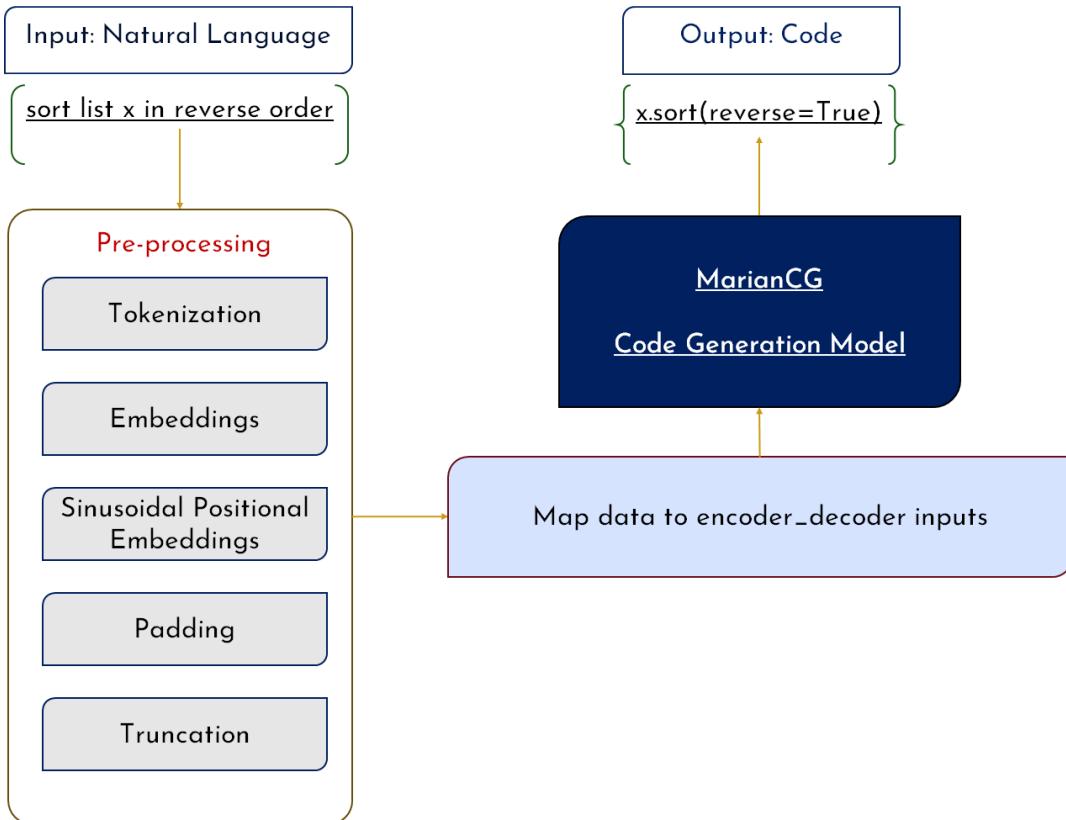


Figure 4.6: MarianCG Encoder Decoder representation for code generation

1.2.1 MarianCG Tokenization

Marian Tokenizer được phát triển và chủ yếu phụ thuộc vào SentencePiece [196]. SentencePiece là mạng nơ-ron tạo văn bản với bộ mã hóa văn bản và bộ giải mã văn bản có kích thước từ vựng trước được xác định trước để đào tạo mô hình nơ-ron. SentencePiece mở rộng đào tạo trực tiếp từ các câu thô để triển khai các đơn vị từ phụ như mô hình ngôn ngữ unigram [197] và mã hóa cặp byte (BPE) [198]. Marian tokenizer được bắt nguồn từ PreTrainedTokenizer trong thư viện biến đổi huggingface, bao gồm phần lớn các phương pháp thiết yếu.

1.2.2 MarianCG Embedding and Positional Embedding

Mô hình MarianCG không chứa mạng nơ-ron tích chập hoặc hồi quy, do đó vai trò của nhúng và nhúng vị trí hiện rất quan trọng và rõ ràng. Vì vậy, bằng cách xác định dữ liệu về vị trí tương đối hoặc tuyệt đối của các mã thông báo trong chuỗi để có thứ tự của chuỗi. Nhúng vị trí và nhúng từ được chia sẻ giữa bộ mã hóa và bộ giải mã. Mô hình MarianCG chứa các nhúng vị trí hình sin cho các nhúng đầu vào tại bộ mã hóa và bộ giải mã.

Nhiệm vụ của nhúng vị trí là cung cấp thông tin về vị trí của từng mã thông báo. Điều này cho phép lớp chú ý tính toán các phản hồi phụ thuộc vào ngữ cảnh, sao cho hai mã thông báo có cùng giá trị trong cụm từ đầu vào nhận được các biểu diễn riêng biệt. Nhúng vị trí hình sin tính toán mã hóa vị trí dưới dạng kết hợp các hàm sin và cos với bước sóng tăng dần theo hình học.

Biểu diễn hình sin hoạt động tốt như biểu diễn đã học và khái quát hóa tốt hơn các chuỗi dài hơn các chuỗi đào tạo. Nó được định nghĩa và chính thức hóa trong bài báo Attention is All You Need [30]. As following, Positional Encoding by Vaswani et al. Let d_{model} be the embedding dimension of words, and $pos \in [0, L-1]$ be the position of a w word in the $w = (w_0, \dots, w_{L-1})$ input sequence. Mathematically, the positional encoding of w is defined in Eq. 4.1.

$$\text{PE}(pos, i) = \begin{cases} \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right), & i = 2k \\ \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right), & i = 2k + 1 \end{cases} \quad (4.1)$$

trong đó mã hóa vị trí tuân theo một mẫu cụ thể, đã học được để xác định vị trí từ hoặc khoảng cách giữa các từ trong chuỗi [199].

MarianCG không có nhúng chuẩn hóa lớp. Vì vậy, nhúng vị trí lấy thứ tự của định danh vị trí được thêm vào các vectơ để bộ biến đổi biết thứ tự của chuỗi.

1.2.3 Marian Encoder and Decoder Architecture

Sau khi phân mã và nhúng với nhúng vị trí, bước tiếp theo là nhập các nhúng này vào Marian Encoder và Marian Decoder. Marian Encoder như thể hiện trong Hình 4.7 là các lớp mã hóa tự chú ý nhiều đầu được kết nối với chuẩn hóa lớp và sau đó, có hai lớp được kết nối đầy đủ và chuẩn hóa lớp cuối cùng.

Khi kiến trúc mã hóa được xây dựng, bộ giải mã có liên quan. Các bước đầu tiên là phân mã và nhúng với nhúng vị trí và bước tiếp theo là nhập các nhúng này vào Marian Encoder và Marian Decoder.

Marian Decoder như thể hiện trong Hình 4.8 có cùng kiến trúc với bộ mã hóa nhưng với việc thêm sự chú ý của bộ mã hóa sau là chuẩn hóa lớp chú ý của bộ mã hóa. Các lớp này có thể được thêm vào trước hai lớp được kết nối đầy đủ.

Thành phần Attention trong mô hình Transformer này thực hiện các phép tính của nó nhiều lần song song. Mỗi thành phần này được gọi là Attention Head. Môđun Attention chia các đối số Query, Key và Value của nó N lần và định tuyến từng phần chia qua một Head riêng biệt. Kết quả của tất cả các thành phần này có thể so sánh được

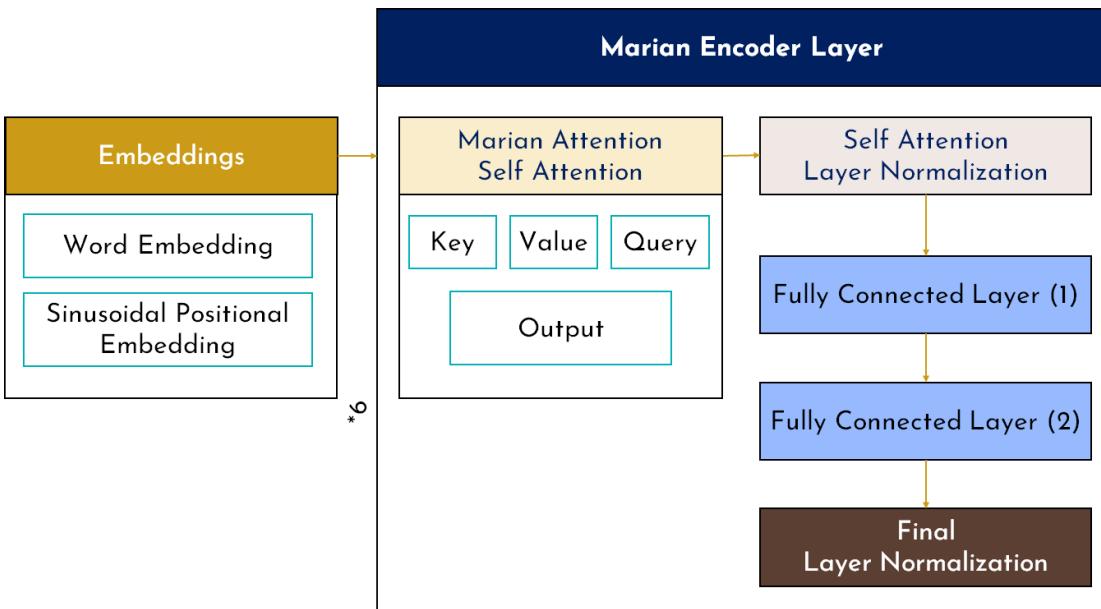


Figure 4.7: Marian Encoder Architecture

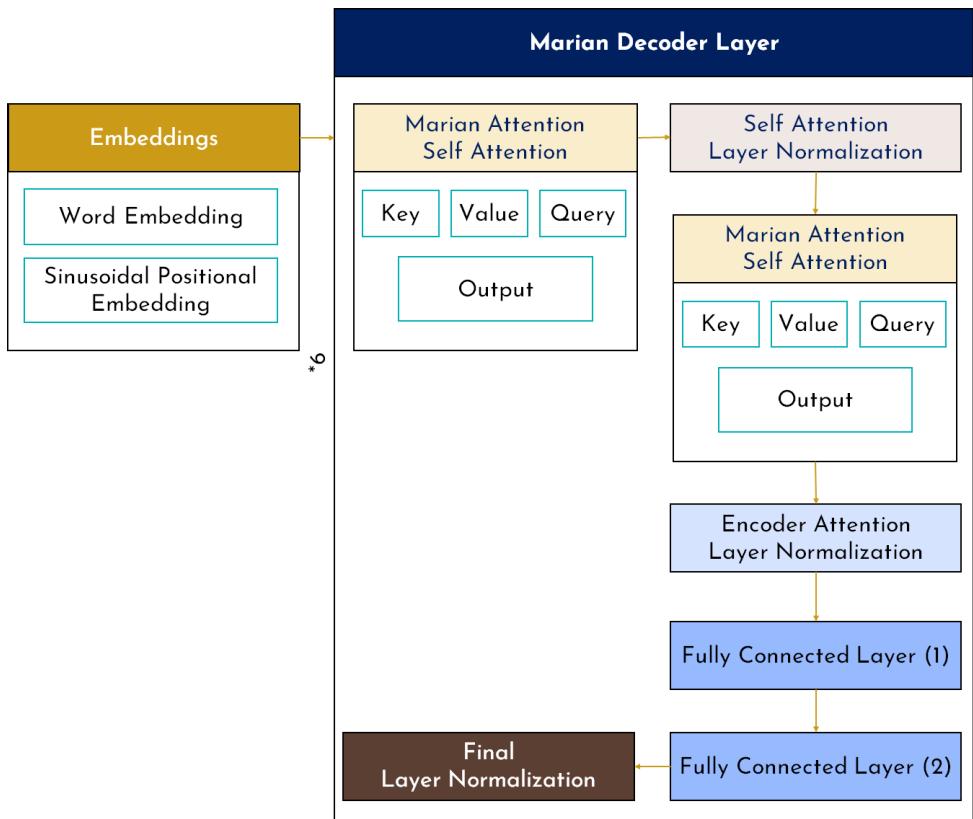


Figure 4.8: Marian Decoder Architecture

Các phép tính chú ý sau đó được cộng lại với nhau để tạo ra Điểm chú ý cuối cùng. Điều này được gọi là sự chú ý nhiều đầu và cho phép Transformer mã hóa nhiều liên kết và sắc thái cho mỗi từ [30].

Có những tính năng chính cho cấu trúc MarianCG mà Marian Tokenizer phụ thuộc vào SentencePiece. Ngoài ra, MarianCG chưa nhúng vị trí hình sin để biểu diễn vị trí của mỗi mã thông báo. Hơn nữa, không có nhúng chuẩn hóa lớp nào cho phương pháp này.

1.3 Proposed Hybrid Models

Tận dụng các mô hình được đào tạo trước cho các tác vụ tạo chuỗi [195], đặc biệt là trong bối cảnh tạo mã, đã trở thành một phương pháp tiếp cận mang tính chuyển đổi trong những năm gần đây. Các mô hình được đào tạo trước là các mô hình ngôn ngữ đã được đào tạo chuyên sâu trên các tập dữ liệu lớn và đa dạng để học các biểu diễn theo ngữ cảnh của ngôn ngữ. Các mô hình này, chẳng hạn như GPT-3 [31], RoBERTa [102] và BERT [60], nắm bắt các mẫu và mối quan hệ phức tạp trong dữ liệu văn bản, giúp chúng thành thạo trong việc hiểu các sắc thái của ngôn ngữ lập trình và cú pháp mã.

Ưu điểm chính của việc sử dụng các mô hình được đào tạo trước để tạo mã nằm ở khả năng khai quát hóa trên nhiều cơ sở mã và ngôn ngữ lập trình khác nhau. Thay vì bắt đầu từ đầu, các mô hình được đào tạo trước đã sở hữu một cơ sở kiến thức đáng kể, cho phép chúng hiểu và tạo mã tốt hơn trong nhiều mô hình lập trình. Sự khai quát hóa này rất quan trọng khi xử lý các tác vụ tạo mã đa ngôn ngữ, vì các mô hình này có thể chuyển đổi liền mạch giữa các ngôn ngữ khác nhau mà không cần đào tạo chuyên sâu về ngôn ngữ.

Một lợi ích đáng kể khác là giảm thời gian đào tạo và yêu cầu về tài nguyên. Các mô hình được đào tạo trước đã được tinh chỉnh trên các tập đoàn dữ liệu lớn, cho phép các nhà nghiên cứu và nhà phát triển chuyển kiến thức này sang các tác vụ tạo mã cụ thể với lượng dữ liệu tương đối nhỏ dành riêng cho từng miền. Việc tinh chỉnh một mô hình được đào tạo trước đòi hỏi ít năng lực tính toán và dữ liệu hơn so với đào tạo từ đầu, giúp cộng đồng rộng lớn dễ tiếp cận hơn.

Hơn nữa, các mô hình được đào tạo trước rất giỏi trong việc nắm bắt các phụ thuộc theo ngữ cảnh và hiểu ngữ cảnh xung quanh khi tạo mã. Nhận thức về ngữ cảnh này giúp tạo ra các đoạn mã mạch lạc và có ý nghĩa về mặt ngữ nghĩa hơn. Các tác vụ tạo mã thường liên quan đến các cấu trúc cú pháp phức tạp, mà các mô hình được đào tạo trước có thể xử lý hiệu quả bằng cách học các mẫu và phụ thuộc phức tạp trong các biểu thức mã.

Tuy nhiên, việc sử dụng các mô hình được đào tạo trước để tạo mã cũng đi kèm với một loạt thách thức. Một mối quan tâm đáng kể là tính an toàn và độ tin cậy của mã được tạo ra. Các mô hình được đào tạo trước đôi khi có thể tạo ra mã không chính xác, không an toàn hoặc không hiệu quả, gây ra rủi ro trong các ứng dụng thực tế. Việc cân nhắc cẩn thận các kỹ thuật xác thực và xác minh là điều cần thiết để đảm bảo tính an toàn và chính xác của mã được tạo ra.

Một thách thức khác là khả năng tiết lộ mã nhạy cảm hoặc sở hữu trí tuệ. Các mô hình được đào tạo trước có thể vô tình ghi nhớ hoặc tiết lộ các đoạn mã bí mật có trong dữ liệu đào tạo của chúng, điều này có thể dẫn đến các vấn đề về quyền riêng tư và bảo mật. Các nhà nghiên cứu và học viên phải sử dụng các kỹ thuật mạnh mẽ để ngăn chặn tình trạng rò rỉ dữ liệu như vậy. Vì vậy, việc tận dụng các mô hình được đào tạo trước cho các tác vụ tạo mã cung cấp một phương pháp tiếp cận mạnh mẽ và hiệu quả để giải quyết sự phức tạp của tác vụ. Ngoài ra, các mô hình này tiếp tục trong nhiều ứng dụng

Khả năng khai quát hóa của các mô hình được đào tạo trên nhiều ngôn ngữ, nhận thức ngữ cảnh và giám yêu cầu đào tạo khiến chúng trở thành một nguồn tài nguyên vô giá để đẩy nhanh tiến độ trong lĩnh vực tạo mã tự động. Tuy nhiên, việc sử dụng có trách nhiệm, cân nhắc về an toàn và các biện pháp bảo vệ quyền riêng tư là rất quan trọng để khai thác hoàn toàn tiềm năng của các mô hình được đào tạo trước để tạo mã và đảm bảo chúng được tích hợp thành công vào quy trình phát triển phần mềm trong thế giới thực.

Chúng tôi đã tận dụng các mô hình được đào tạo trước để có được nhiều mô hình tạo mã khác nhau với các mô hình được đào tạo trước khác nhau với các bộ mã hóa và bộ giải mã Marian khác nhau như thể hiện trong Hình 4.9. Các mô hình đề xuất của chúng tôi được chỉ ra với các bộ mã hóa và bộ giải mã của chúng trong Bảng 4.1.

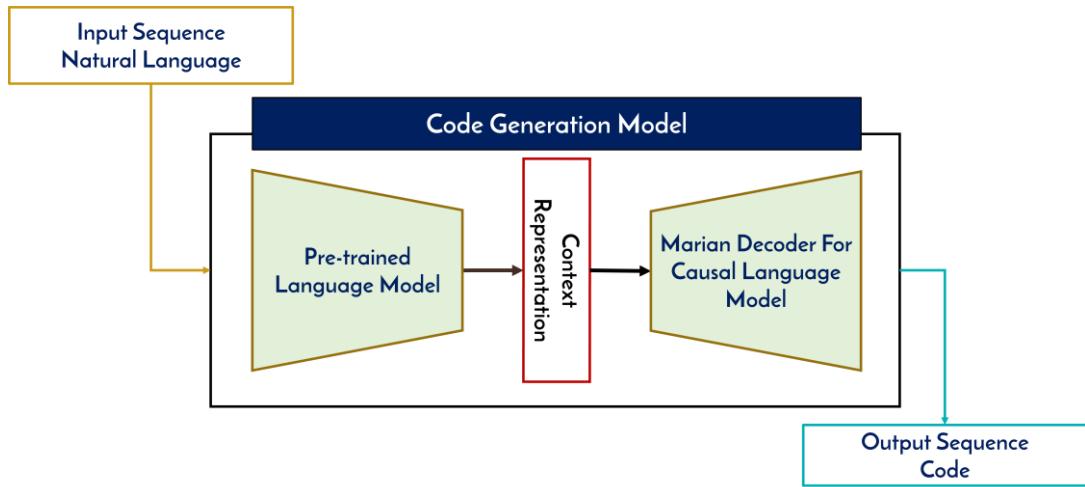


Figure 4.9: Leveraging Pre-trained Language models for building Code Generation Models

1.3.1 RoBERTaMarian Model

Năm 2019, mô hình RoBERTa [33] được Facebook giới thiệu và dựa trên mô hình được đào tạo trước Google BERT [7] được đề xuất trước đó vào năm 2019. Mô hình này được xây dựng trên mô hình BERT bằng cách loại bỏ mục tiêu đào tạo trước câu tiếp theo và đào tạo với các lô nhỏ và tốc độ học lớn hơn đáng kể. Mô hình này giống hệt với Mô hình BERT, ngoại trừ một điều chỉnh

nhưng nhỏ và thiết lập cho các mô hình được đào tạo trước RoBERTa.

RoBERTa được xây dựng trên cùng một kiến trúc với BERT, nhưng nó sử dụng BPE cấp byte làm trình phân tích cú pháp (tương tự như GPT-2) và một chiến lược đào tạo trước khác.

Table 4.1: Our proposed Code Generation Models

No.	Model	Model Architecture	
		Encoder Model	Decoder Model
1	MarianCG Model	Marian Encoder	Marian Decoder
2	RoBERTaMarian	DistilRoBERTa	Marian Decoder
3	BERTMarian	DistilBERT	Marian Decoder
4	ELECTRAMarian	ELECTRA	Marian Decoder
5	LUKEMarian	LUKE	Marian Decoder

Mã loại mã thông báo không tồn tại trong mô hình RoBERTa. Kiến trúc của mô hình cơ sở RoBERTa có 12 lớp. RoBERTa được đào tạo trước với tác vụ MLM (và không có tác vụ NSP).

Chúng tôi đã sử dụng phiên bản chung cất của mô hình cơ sở RoBERTa được gọi là Distil-RoBERTa và có quy trình đào tạo là DistilBERT [200]. Mô hình DistilRoBERTa được đào tạo trước có 6 lớp, 768 chiều và 12 đầu, với tổng cộng 82 triệu tham số (so với 125 triệu tham số của RoBERTa-base). Trung bình, DistilRoBERTa nhanh gấp đôi so với mô hình cơ sở Roberta. Mô hình DistilRoBERTa phân biệt giữa tiếng Anh và tiếng Anh. Đây là mô hình phân biệt chữ hoa chữ thường. Chúng tôi đã kết hợp DistilRoBERTa làm bộ mã hóa với Marian Decoder như thể hiện trong Hình 4.10 và nhận được kết quả là 35,74 cho điểm BLEU và 44,25 cho điểm ROUGE trên tập dữ liệu CoNaLa.

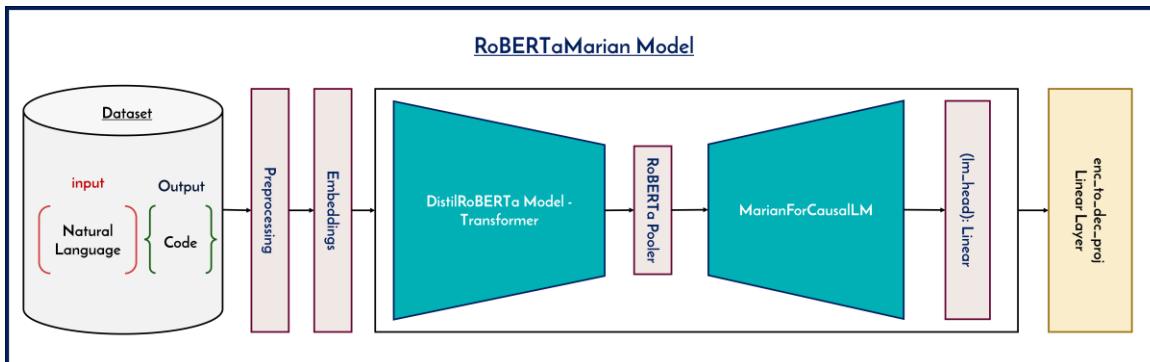


Figure 4.10: RoBERTaMarian Code Generation Model

Các thay đổi siêu tham số do RoBERTa thực hiện so với BERT như sau:

- Đào tạo mô hình trong thời gian dài hơn, với các đợt lớn hơn và nhiều dữ liệu hơn.
- Xóa mục tiêu dự đoán câu tiếp theo.
- Dữ liệu đào tạo lớn hơn (x10, từ 16G đến 160GB) và kích thước đợt lớn hơn (từ 256 đến 8k).
- Kích thước từ vựng lớn hơn (từ 30k đến 50k).
- Các chuỗi dài hơn được sử dụng làm đầu vào (nhưng vẫn giữ nguyên giới hạn 512 mã thông báo).
- Che giấu động. Thay đổi mẫu che giấu được sử dụng cho dữ liệu đào tạo một cách động.

DistilRoBERTa được đào tạo bằng OpenWebTextCorpus, một bản sao tập dữ liệu OpenAI WebText. Tập dữ liệu này nhỏ hơn bốn lần so với tập dữ liệu được sử dụng để đào tạo mô hình giáo viên RoBERTa. Hình 4.11 cho thấy kiến trúc của mô hình RoBERTaMarian và kiến trúc bộ mã hóa được khai báo và hiển thị trong Hình 4.12

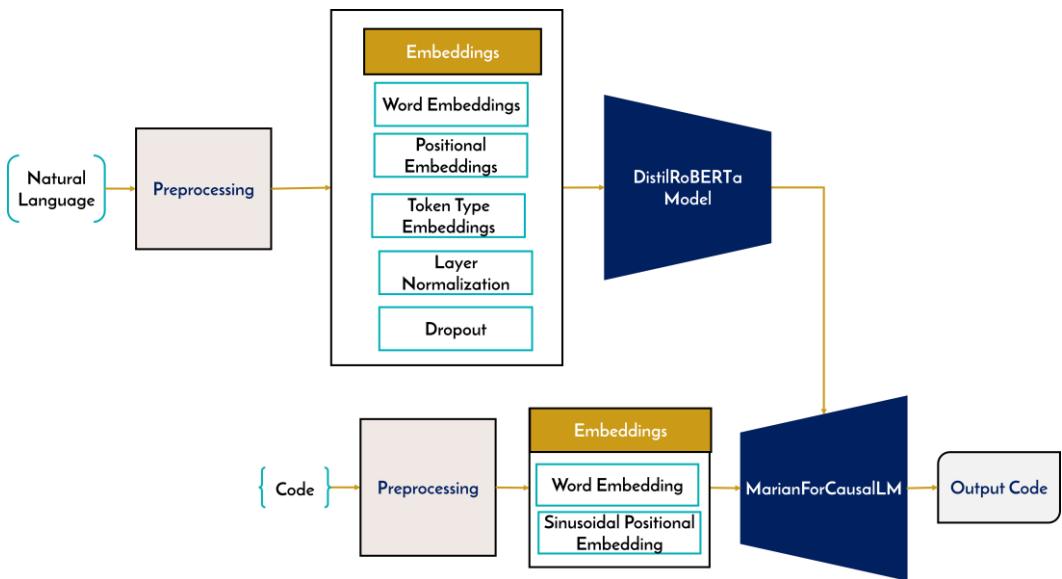


Figure 4.11: RoBERTaMarian Model Architecture

1.3.1.1 RoBERTa Pooler Layer

Thuật ngữ "Roberta Pooler" liên quan đến lớp chuyên biệt trong mô hình RoBERTa được gọi là lớp pooler. RoBERTa đại diện cho sự tinh chỉnh của mô hình BERT (Bidirectional Encoder Representations from Transformers), một khuôn khổ được sử dụng rộng rãi cho các tác vụ liên quan đến Xử lý dựa trên Transformer.

Trong kiến trúc RoBERTa, lớp pooler chịu trách nhiệm cô đọng thông tin thu thập được từ các lớp mã hóa thành một biểu diễn nhất quán và được xác định trước. Biểu diễn kết quả này chứng tỏ có giá trị đối với các tác vụ tiếp theo như

phân loại hoặc tạo văn bản. Điều này đạt được bằng cách xử lý các trạng thái ẩn của lớp cuối cùng làm đầu vào và sau đó áp dụng một hoạt động gộp, thường liên quan đến các kỹ thuật gộp trung bình hoặc gộp tối đa, để có được một biểu diễn vecto kỳ dị.

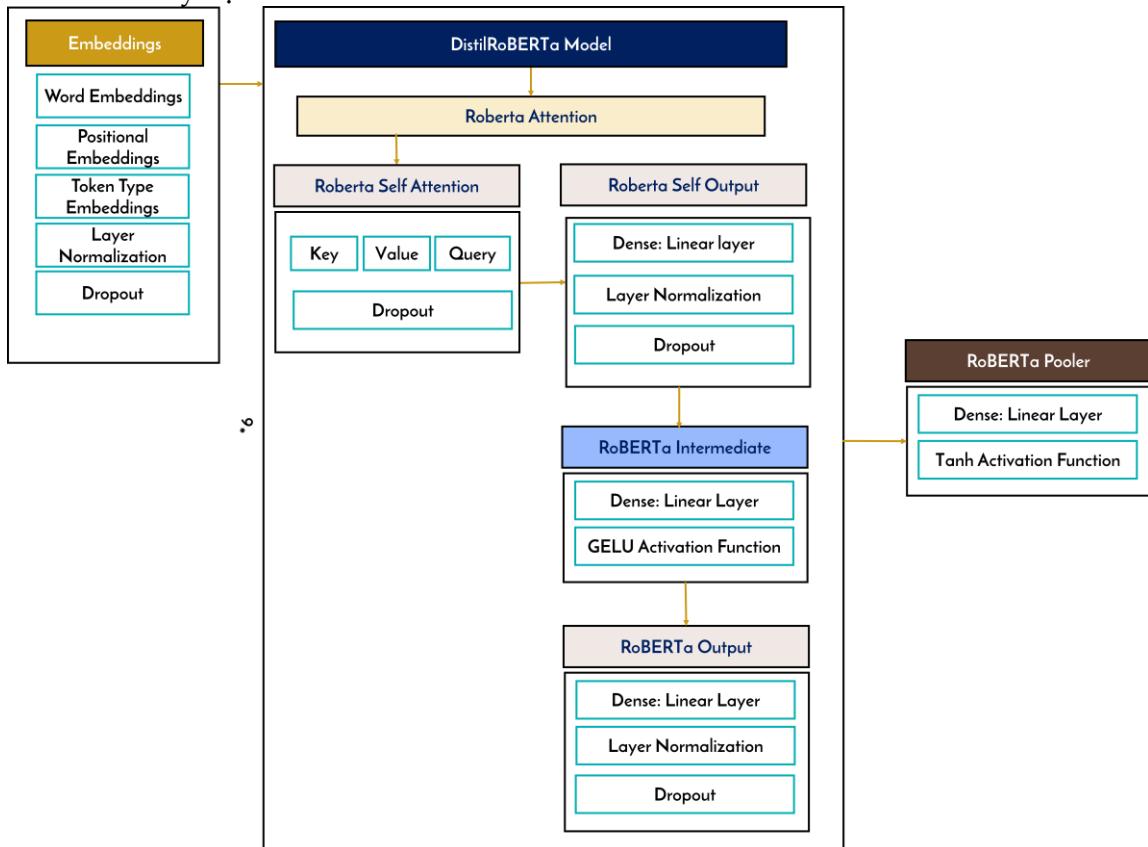


Figure 4.12: RoBERTaMarian Encoder

au đó, vecto này phù hợp để sử dụng trong lớp phân loại hoặc giải mã để tạo điều kiện cho dự đoán hoặc tạo văn bản.

Lớp pooler trong RoBERTa cấu thành một lớp được kết nối đầy đủ được trang bị các trọng số và độ lệch có thể học được. Các tham số này được tối ưu hóa trong giai đoạn đào tạo của mô hình để nắm bắt hiệu quả các chi tiết nổi bật từ chuỗi đầu vào và tạo ra một biểu diễn có mục đích phục vụ đúng cho nhiệm vụ trong tầm tay.

Thông thường, quyền truy cập vào lớp pooler được tạo điều kiện thông qua thuộc tính 'pooler_output' được nhúng trong đầu ra của mô hình RoBERTa. Thuộc tính này giữ kết quả của các hoạt động của lớp pooler, biểu hiện dưới dạng một tensor được đặc trưng bởi các chiều (batch_size, hidden_size). Trong ngữ cảnh này, 'batch_size' biểu thị số lượng chuỗi đầu vào trong một lô, trong khi 'hidden_size' biểu thị phạm vi chiều của cả lớp mã hóa và lớp pooler.

Điều đáng chú ý là việc kết hợp lớp pooler trong RoBERTa không phải là bắt buộc và một số biến thể hoặc triển khai nhất định có thể chọn bỏ qua nó. Tuy nhiên, trong phần lớn các trường hợp, lớp pooler được chấp nhận do tiện ích thực tế của nó trong việc tạo ra biểu diễn chuẩn hóa của chuỗi đầu vào.

1.3.2 BERTMarian Model

DistilBERT là phiên bản BERT dành cho học sinh như thể hiện trong Hình 4.13. Nó nhỏ hơn và nhanh hơn BERT. Mô hình biến đổi DistilBERT được tạo ra bằng cùng thiết kế cơ bản như BERT. Các nhúng kiểu pooler và token bị loại bỏ và số lớp giám đi gấp đôi, tạo ra bộ mã hóa và giải mã nhỏ hơn với sáu lớp.

DistilBERT đã chứng minh rằng các biến thể trong chiều cuối cùng của tenxơ (chiều kích thước ẩn) có tác động nhỏ hơn đến hiệu quả tính toán (đối với ngân sách tham số cố định) so với các biến thể trong các yếu tố khác như số lớp.

Do đó, DistilBERT có mục đích tập trung vào việc giảm số lớp. Nó được tự giám sát và đào tạo trước trên cùng một ngũ liệu như một giáo viên sử dụng mô hình cơ sở BERT. Nó sử dụng một quy trình tự động để tạo đầu vào và nhãn từ các văn bản đó bằng mô hình cơ sở BERT. DistilBERT có ít hơn 40% tham số so với bert-base-uncased DistilBERT có ít hơn 40% tham số so với bert-base-uncased. Nó chạy nhanh hơn 60% trong khi vẫn duy trì hơn 95% hiệu suất của BERT trên chuẩn mực kiểm tra hiểu ngôn ngữ GLUE.

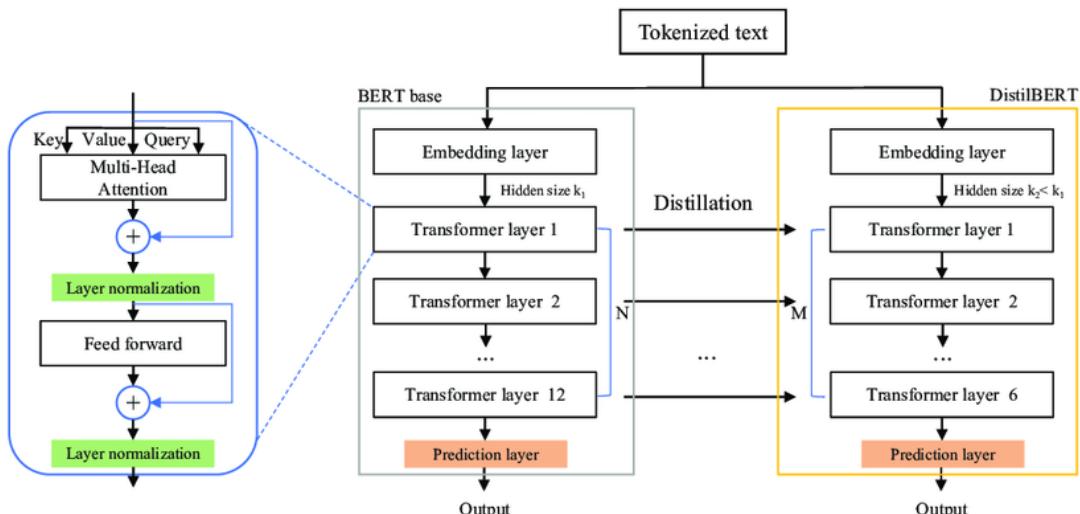


Figure 4.13: DistilBERT Model Architecture[200]

Chúng tôi đã xây dựng một mô hình tạo mã như thể hiện trong Hình 4.14 có chứa DistilBERT làm bộ mã hóa và Marian Decoder và được đào tạo trên tập dữ liệu CoNaLa. Mô hình này có kết quả trên tập dữ liệu CoNaLa là 32,46 điểm BLEU và 43,95 điểm ROUGE trên tập dữ liệu CoNaLa.

Hình 4.15 cho thấy kiến trúc của mô hình BERTMarian và kiến trúc bộ mã hóa được khai báo và thể hiện trong Hình 4.16.

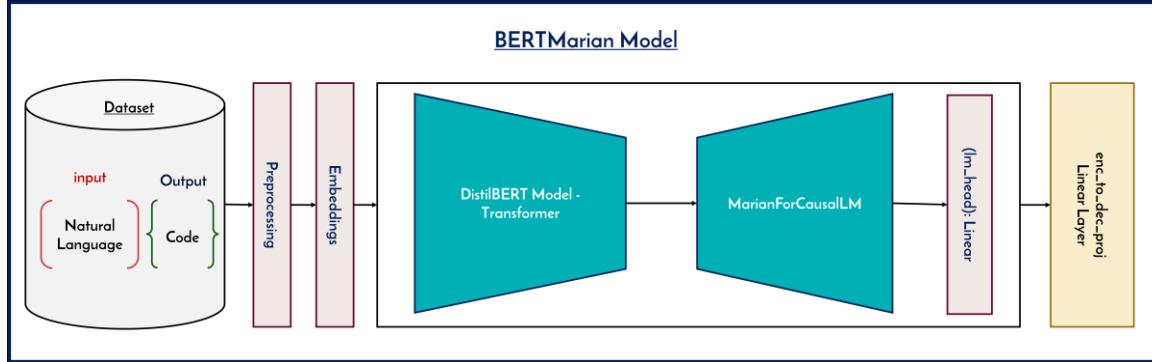


Figure 4.14: BERTMarian Code Generation Model

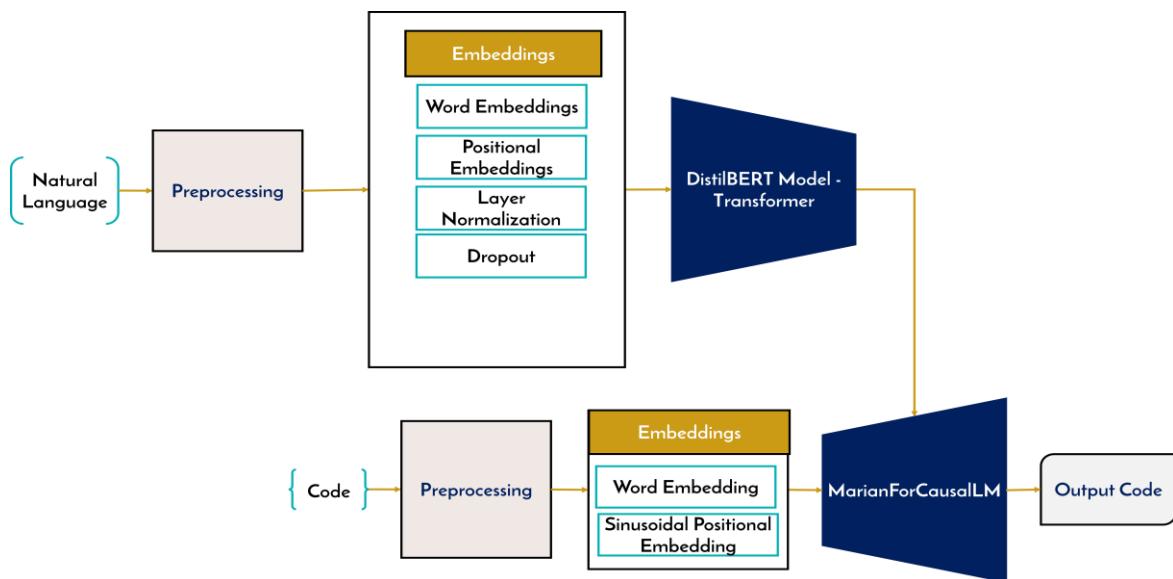


Figure 4.15: BERTMarian Model Architecture

1.3.3 ELECTRAMarian Model

Năm 2020, mô hình ELECTRA [201] được Đại học Stanford giới thiệu với sự hợp tác của Google và đây là một phương pháp mới để học các biểu diễn ngôn ngữ tự giám sát. Nó được sử dụng để đào tạo trước các mạng lưới biến áp với một lượng tính toán nhỏ. Các mô hình ELECTRA, giống như các bộ phân biệt GAN, được đào tạo để xác định các mã thông báo đầu vào "thực" từ các mã thông báo đầu vào "giả" do một mạng no-ron khác tạo ra như thể hiện trong Hình 4.17. ELECTRA cung cấp các kết quả ẩn tượng ở quy mô khiêm tốn. Trên tập dữ liệu SQuAD 2.0, ELECTRA tạo ra các kết quả tiên tiến ở quy mô lớn.

Mô hình đào tạo trước này hiệu quả hơn Mô hình ngôn ngữ bị che giấu (MLM) vì nó được xác định trên tất cả các mã thông báo đầu vào thay vì chỉ là phần lựa chọn nhỏ bị che giấu. Với cùng kích thước mô hình, dữ liệu và tính toán, các biểu diễn theo ngữ cảnh mà mô hình ELECTRA thu được vượt trội

hơn các biểu diễn được đào tạo bởi BERT. Những cải tiến này đặc biệt đáng kể đối với các mô hình nhỏ.

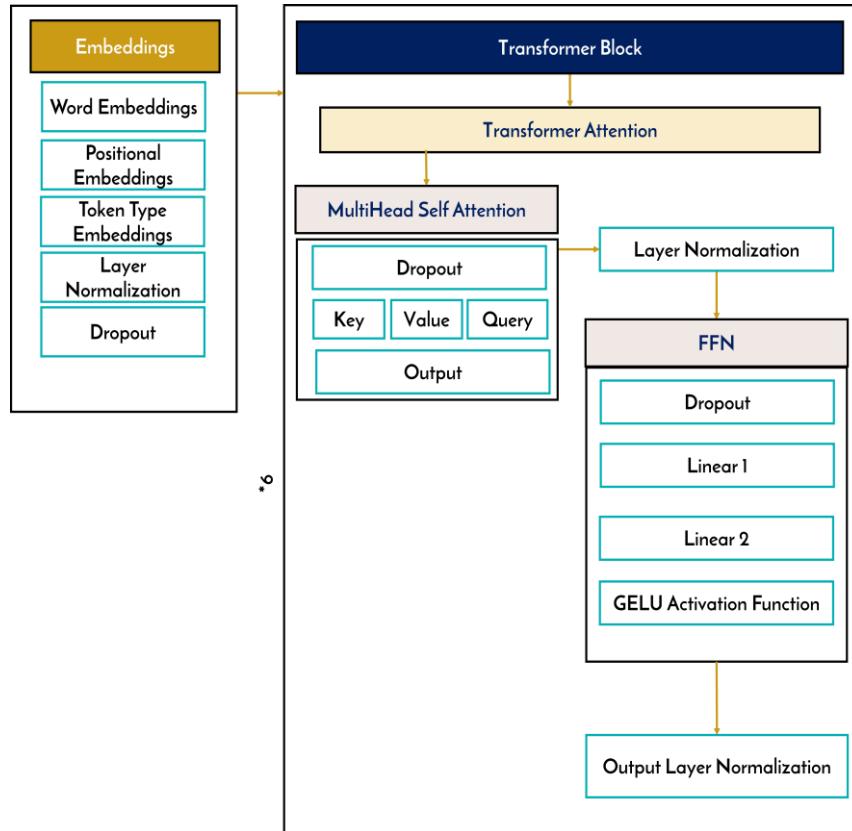


Figure 4.16: BERTMarian Encoder

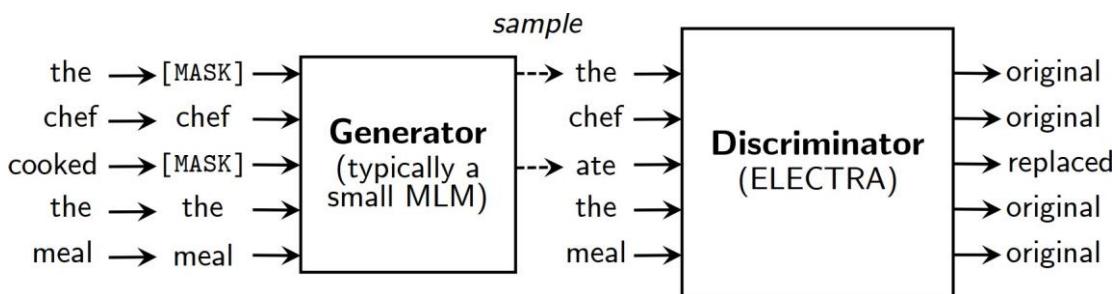


Figure 4.17: ELECTRA model like GAN[201]

Chúng tôi đã sử dụng google electra-base-discriminator có 6 lớp ẩn như thể hiện trong Hình 4.18. Mô hình này như một bộ mã hóa ELECTRA với Marian Decoder đã tạo ra kết quả là 30,18 cho số liệu điểm BLEU và 42,42 điểm ROUGE trên CoNaLa.

Hình 4.19 cho thấy kiến trúc của mô hình ELECTRAMarian và kiến trúc bộ mã hóa được khai báo và thể hiện trong Hình 4.20.

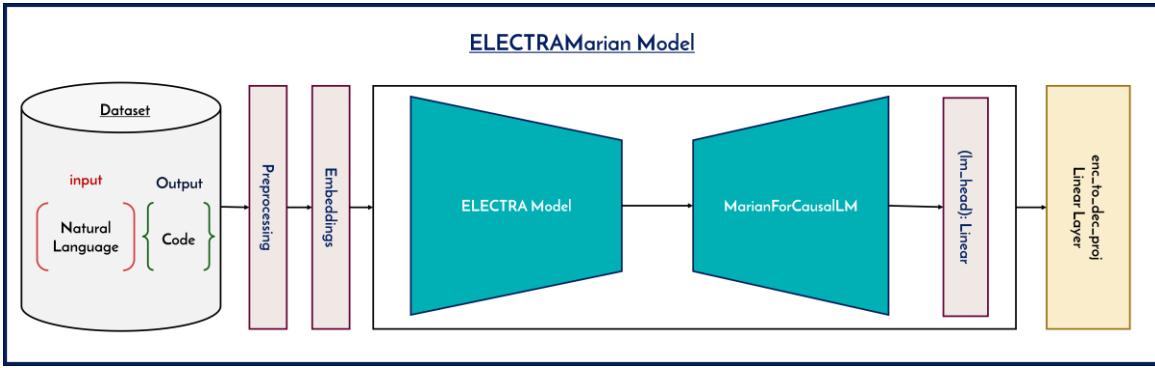


Figure 4.18: ELECTRAMarian Code Generation Model

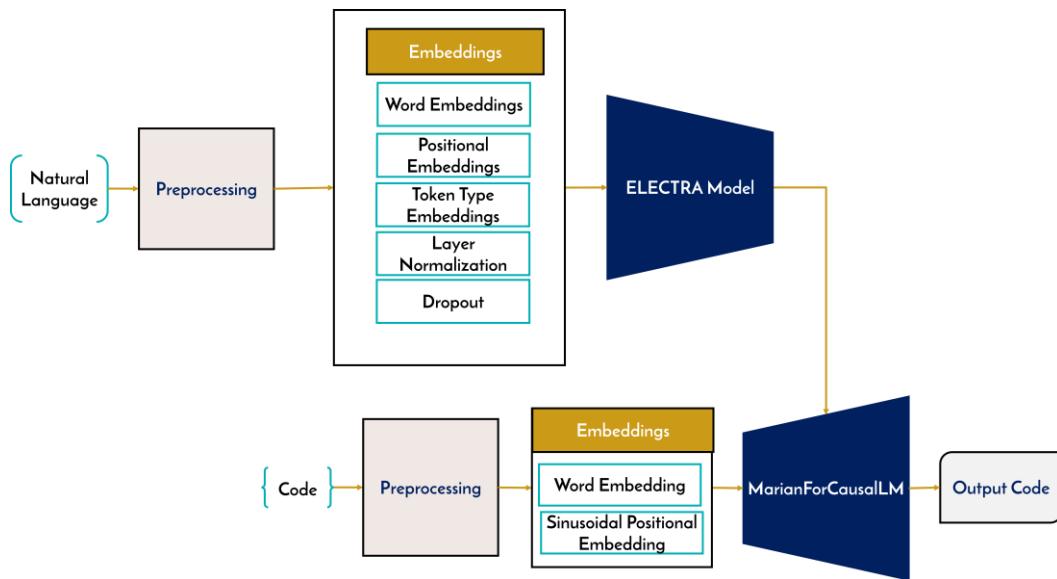


Figure 4.19: ELECTRAMarian Model Architecture

1.3.4 LUKEMarian Model

LUKE [202] là một mô hình biến đổi được đào tạo bằng cách sử dụng một lượng lớn dữ liệu chủ thích thực thể thu thập được từ Wikipedia. Nó xử lý không chỉ các từ mà còn cả các thực thể như các mã thông báo riêng biệt và sử dụng bộ biến đổi để xây dựng các biểu diễn trung gian và đầu ra cho tất cả các mã thông báo. LUKE khác với các biểu diễn từ ngữ cảnh hóa trước đó (CWR) và có thể mô phỏng trực tiếp các mối quan hệ thực thể vì các thực thể được biểu diễn dưới dạng mã thông báo như thể hiện trong Hình.4.21.

LUKE được đào tạo bằng một tác vụ tiền đào tạo mới là phần mở rộng đơn giản của mô hình ngôn ngữ bị che giấu (MLM) của BERT. Công việc này bao gồm việc che giấu các thực thể một cách ngẫu nhiên bằng cách thay thế chúng bằng các thực thể [MASK] và đào tạo mô hình bằng cách dự đoán các thực thể gốc của các thực thể bị che giấu này. RoBERTa được sử dụng làm mô hình tiền đào tạo cơ bản và đào tạo trước mô hình bằng cách đồng thời tối đa hóa các mục tiêu MLM. Như thể hiện trong Hình.4.22, chúng tôi đã kết hợp mô hình LUKE với bộ giải mã Marian để xây dựng mô hình lai Seq2Seq cuối cùng của

mình. Ý tưởng này có một số lợi thế để biểu diễn các từ theo giá trị và thực thể của chúng.

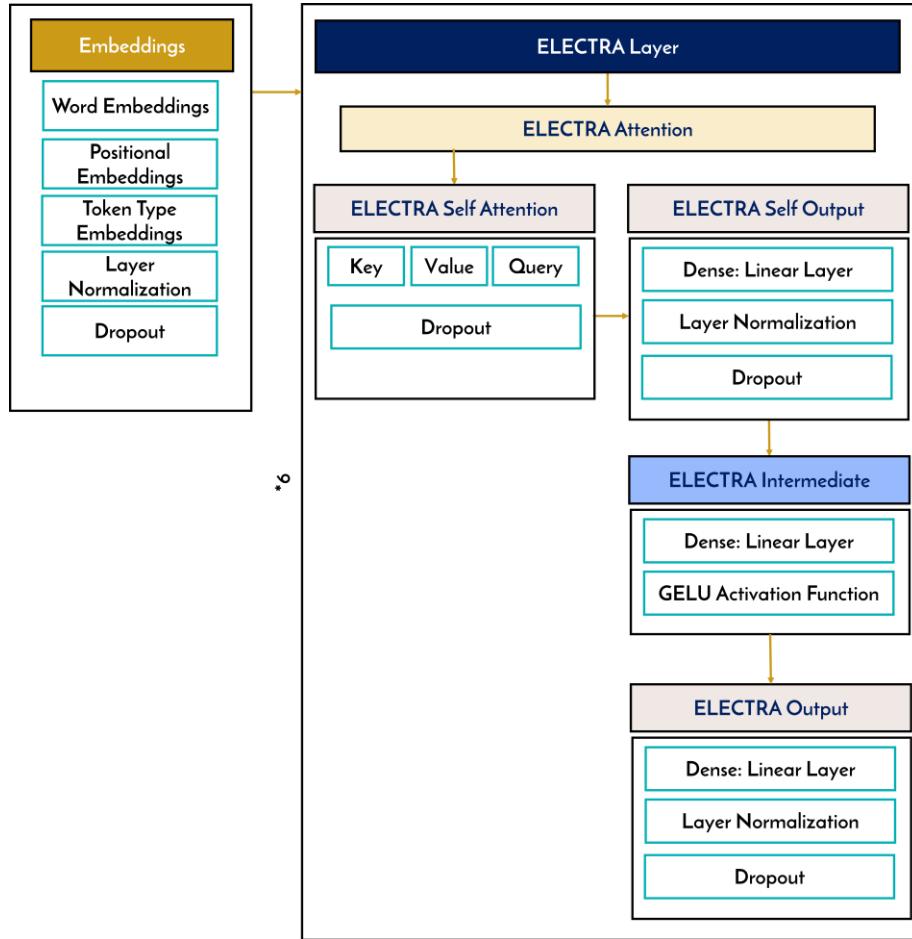


Figure 4.20: ELECTRAMarian Encoder

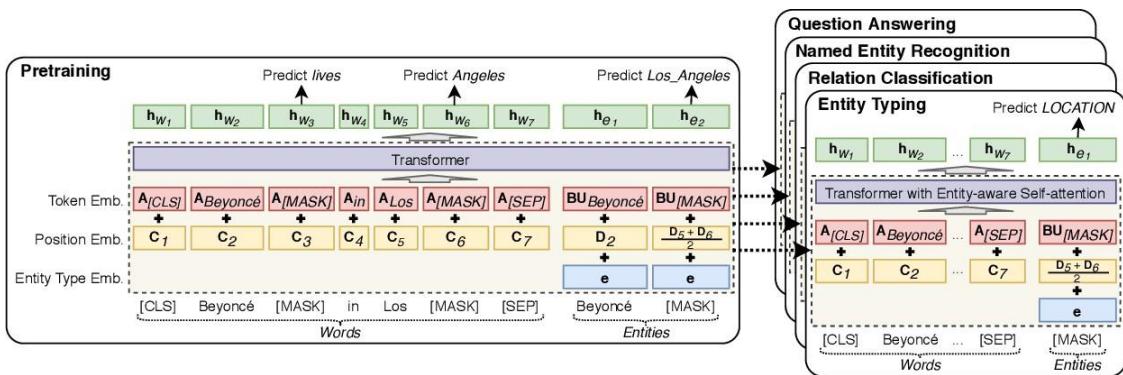


Figure 4.21: LUKE outputs are contextualized representations for each word and entity[202]

LUKE là một biểu diễn ngữ cảnh được tạo ra chủ yếu cho các hoạt động liên quan đến thực thể. Sử dụng một lượng lớn ngữ liệu chú thích thực thể có được từ Wikipedia, LUKE được đào tạo để dự đoán các từ và thực thể được che giấu ngẫu nhiên. LUKE biểu diễn từ và thực thể của nó. Mô hình cơ sở LUKE có 12

lớp ẩn và kích thước ẩn bằng 768. Nó có tổng số 253M tham số.

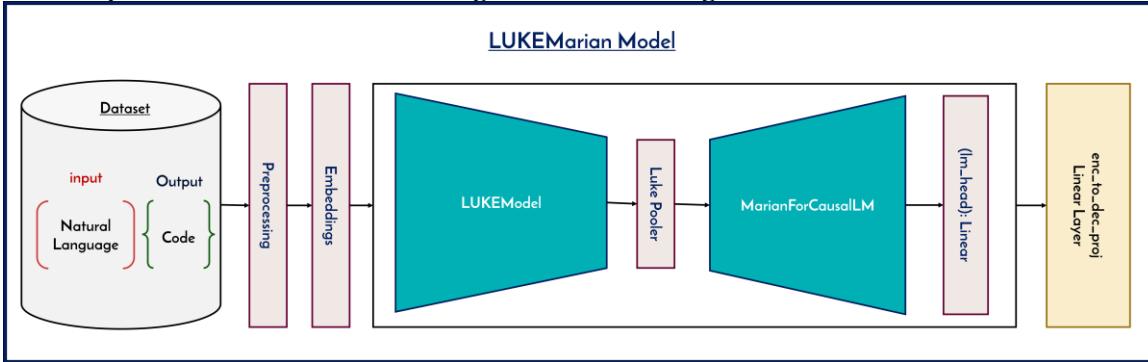


Figure 4.22: LUKE Marian Code Generation Model

Khi LUKE được áp dụng cho các tác vụ hạ lưu, nó tính toán các biểu diễn của các thực thể tùy ý trong văn bản bằng cách sử dụng các thực thể [MASK] làm đầu vào. Nếu tác vụ bao gồm chủ thích thực thể, mô hình sẽ tạo các biểu diễn thực thể dựa trên thông tin tập trung vào thực thể phong phú được lưu trữ trong các nhúng thực thể có liên quan. Chúng tôi đã sử dụng mô hình cơ sở LUKE làm bộ mã hóa nhưng chỉ cách nó 6 lớp. Kết hợp mô hình cơ sở LUKE 6 lớp này với Marian Decoder và đào tạo trên tập dữ liệu CoNaLa đã thu được kết quả 29,83 cho điểm BLEU và 39,84 điểm Rouge cho dữ liệu thử nghiệm.

Hình 4.23 cho thấy kiến trúc của mô hình LUKE Marian và kiến trúc bộ mã hóa được khai báo và hiển thị trong Hình 4.24.

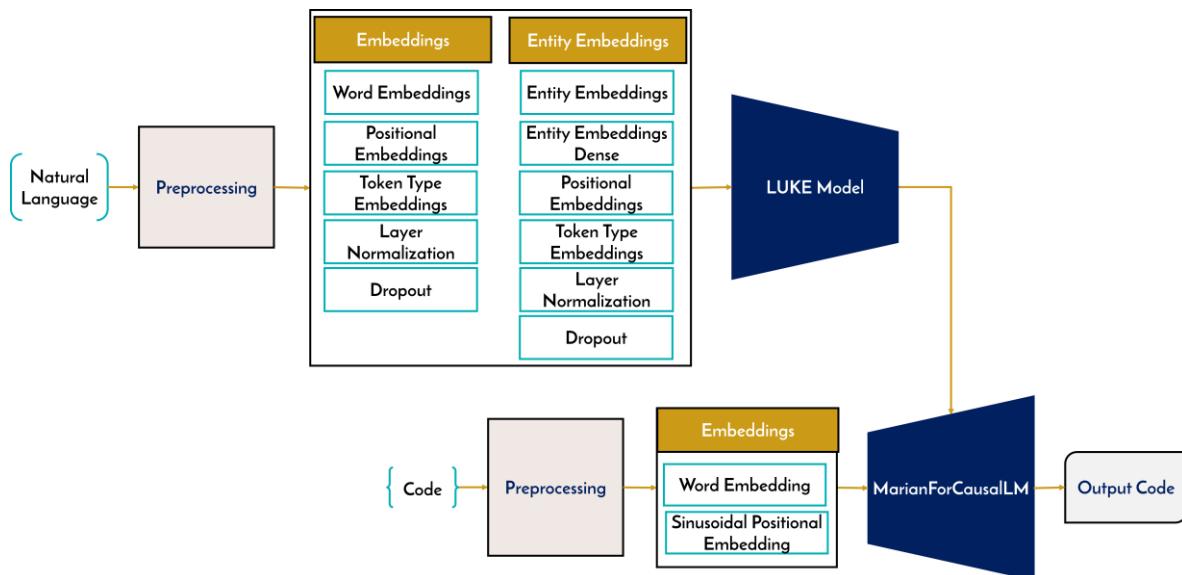


Figure 4.23: LUKE Marian Model Architecture

1.3.4.1 LUKE Pooler Layer

Lớp pooler trong mô hình LUKE (LUKE là viết tắt của "Hiểu ngôn ngữ với nhúng dựa trên kiến thức") đóng vai trò là lớp tóm tắt, lấy các biểu diễn theo ngữ cảnh của các mã thông báo đầu vào và tạo ra một biểu diễn có độ dài cố định, thường được gọi

là biểu diễn gộp hoặc biểu diễn cấp câu..

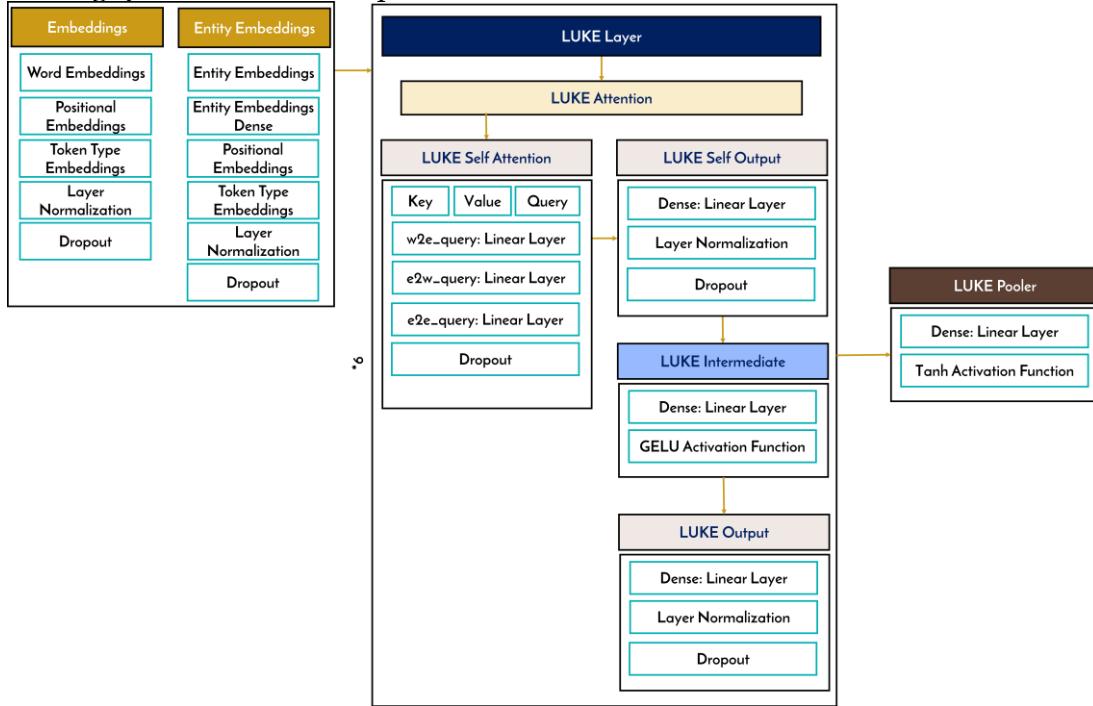


Figure 4.24: LUKE Marian Encoder

Chức năng cụ thể của lớp pooler trong LUKE bao gồm những chức năng sau:

- Tổng hợp các biểu diễn mã thông báo: Lớp pooler lấy các biểu diễn ngữ cảnh của các mã thông báo đầu vào, thường lấy từ các lớp mã hóa biến áp và tổng hợp chúng thành một biểu diễn duy nhất. Quá trình tổng hợp này tóm tắt thông tin từ các mã thông báo riêng lẻ và nắm bắt ý nghĩa chung của chuỗi đầu vào.
- Mã hóa cấp câu: Biểu diễn gộp do lớp pooler tạo ra sẽ đóng gói thông tin ngữ cảnh từ toàn bộ chuỗi đầu vào. Biểu diễn này thường được sử dụng làm mã hóa cấp câu có thể đưa vào các tác vụ hạ lưu, chẳng hạn như nhận dạng thực thể hoặc trích xuất quan hệ.
- Trích xuất ngữ nghĩa: Lớp pooler giúp trích xuất thông tin ngữ nghĩa cấp cao từ các mã thông báo đầu vào. Bằng cách tóm tắt các biểu diễn mã thông báo, lớp này tập trung vào việc nắm bắt các tính năng nổi bật nhất của chuỗi đầu vào và loại bỏ thông tin ít liên quan hoặc nhiễu.

Nhìn chung, lớp pooler trong LUKE đóng vai trò quan trọng trong việc tạo ra một biểu diễn có độ dài cố định nắm bắt thông tin theo ngữ cảnh và ý nghĩa ngữ nghĩa của chuỗi đầu vào. Biểu diễn pooled này sau đó có thể được sử dụng cho nhiều tác vụ hạ nguồn khác nhau đòi hỏi phải hiểu ở cấp độ câu hoặc trích xuất kiến thức.

1.4 Error Analysis and Correction

Mã hóa hỗ trợ AI là một lĩnh vực phát triển nhanh chóng, kết hợp sức mạnh của trí tuệ nhân tạo (AI) và máy học với chuyên môn của các lập trình viên con người để nâng cao và hợp lý hóa quy trình phát triển phần mềm. Công nghệ này tận dụng các thuật toán thông minh và mô hình dữ liệu để hỗ trợ các nhà phát triển viết mã, cải thiện chất lượng mã và tăng năng suất chung. Phát triển phần mềm truyền thống đòi hỏi nhiều nỗ lực thủ công để viết và gỡ lỗi mã, thường dẫn đến các quy trình tốn thời gian và dễ xảy ra lỗi.

Mã hóa hỗ trợ AI nhằm mục đích giảm bớt những thách thức này bằng cách cung cấp các công cụ tự động phân tích các cơ sở mã hiện có, hiểu các mẫu lập trình và tạo ra các đề xuất hoặc hoàn thành các phần mã. Khái niệm cơ bản đầu tiên sau mã hóa hỗ trợ AI liên quan đến việc đào tạo các mô hình máy học trên một lượng lớn kho lưu trữ mã để tìm hiểu các mẫu, cú pháp và các phương pháp hay nhất. Sau đó, các mô hình này có thể hỗ trợ các nhà phát triển bằng cách cung cấp các đề xuất hoàn thiện mã, xác định các lỗi hoặc lỗ hổng tiềm ẩn và cung cấp các tối ưu hóa dựa trên các tiêu chuẩn mã hóa đã thiết lập.

Một trong những lợi thế chính của mã hóa hỗ trợ AI là khả năng nâng cao năng suất của nhà phát triển. Bằng cách tự động hóa các tác vụ lặp đi lặp lại và cung cấp các đề xuất thông minh, các nhà phát triển có thể tập trung vào các quyết định thiết kế cấp cao hơn và giải quyết vấn đề, đẩy nhanh quá trình phát triển tổng thể. Hơn nữa, trợ lý mã có thể cải thiện đáng kể chất lượng mã. Bằng cách phân tích một lượng lớn mã, thuật toán AI có thể xác định các lỗi mã hóa phổ biến, phát hiện các lỗi tiềm ẩn và đề xuất tái cấu trúc hoặc tối ưu hóa mã. Điều này giúp giảm lỗi và tăng cường độ tin cậy, khả năng bảo trì và hiệu suất của các ứng dụng phần mềm.

Một lợi ích khác của trợ lý mã là khả năng tạo điều kiện thuận lợi cho việc chuyển giao kiến thức và chia sẻ kỹ năng trong các nhóm phát triển. Bằng cách nắm bắt và phân tích các mẫu mã hóa, các phương pháp hay nhất và triển khai mã thành công, các công cụ mã hóa hỗ trợ AI có thể hỗ trợ các nhà phát triển ít kinh nghiệm hơn trong việc học hỏi từ chuyên môn chung của các đồng nghiệp, cuối cùng là nâng cao trình độ kỹ năng của toàn bộ nhóm. Tuy nhiên, điều quan trọng cần lưu ý là mã hóa hỗ trợ AI không nhằm mục đích thay thế các lập trình viên con người. Thay vào đó, nó đóng vai trò là một công cụ mạnh mẽ để tăng cường khả năng của họ, cải thiện hiệu quả và cho phép họ giải quyết các tác vụ mã hóa phức tạp hiệu quả hơn.

1.4.1 Error Analysis in the Generated Code

Tạo mã là một lĩnh vực nghiên cứu và phát triển thú vị nhằm mục đích thu hẹp khoảng cách giữa ngôn ngữ con người và ngôn ngữ lập trình. Trong nhiệm vụ này, đầu vào là mô tả và thông số kỹ thuật của con người, và đầu ra là mã Python. Để thực hiện điều này, chúng tôi đã triển khai các mô hình được thiết kế riêng cho nhiệm vụ tạo mã. Tuy nhiên, việc tạo mã tự động đôi khi có thể dẫn đến các vấn đề về linting và lỗi trong mã đã tạo. Để giải quyết vấn đề này,

chúng tôi đã sử dụng nhiều công cụ và kỹ thuật khác nhau để đảm bảo mã đã tạo đáp ứng các tiêu chuẩn bắt buộc.

Đầu tiên, chúng tôi sử dụng Flake8², một trình kiểm tra lỗi Python phổ biến, để phân tích mã được tạo ra và xác định bất kỳ vấn đề tiềm ẩn nào. Flake8 kiểm tra mã để tìm các vi phạm hướng dẫn về phong cách và lỗi phổ biến, cung cấp phản hồi về các lĩnh vực cần cải thiện. Ngoài ra, chúng tôi đã thực hiện phân tích cú pháp trên mã được tạo ra để xác minh tính chính xác của nó như là mã Python. Phân tích này bao gồm kiểm tra thực lề đúng, sử dụng cú pháp đúng và tuân thủ chung các quy tắc ngôn ngữ Python. Bất kỳ sửa đổi nào cần thiết để làm cho mã phù hợp như mã Python hợp lệ đều được triển khai.

1.4.2 Code Refining and Correction

Chúng tôi đã sử dụng một số công cụ và thư viện sửa lỗi để nâng cao hơn nữa chất lượng và khả năng đọc của mã được tạo. Một trong những công cụ như vậy là Autopep8³, tự động định dạng mã Python theo các hướng dẫn về kiểu đã xác định. Công cụ này có thể sửa các sự cố liên quan đến thực lề, độ dài dòng và khoảng trắng, giúp mã sạch hơn và nhất quán hơn.

Chúng tôi cũng đã sử dụng thư viện Add-trailing-comma4, công cụ này thêm dấu phẩy theo sau vào danh sách và từ điển Python. Dấu phẩy theo sau có thể cải thiện khả năng đọc mã và giúp ngăn ngừa lỗi, đặc biệt là khi sửa đổi hoặc mở rộng các cấu trúc dữ liệu này. Để định dạng mã nhất quán, chúng tôi đã sử dụng Yapf5 và Black6, cả hai đều là trình định dạng mã Python phổ biến. Các công cụ này tự động định dạng mã theo nhiều hướng dẫn về kiểu khác nhau, chẳng hạn như PEP 8 và hướng dẫn về kiểu của Google. Bằng cách áp dụng các trình định dạng này, mã được tạo ra sẽ dễ đọc và bảo trì hơn.

YAPF là trình định dạng cho các tệp Python do Google phát triển. Trình định dạng này được thiết kế để có khả năng cấu hình cao và ít ảnh hưởng đến mã đang được định dạng. YAPF định dạng lại mã để tuân theo hướng dẫn về kiểu được chỉ định trong PEP 8, nhưng tập trung vào khả năng đọc và định dạng nhất quán. YAPF có sẵn để cài đặt thông qua pip và có thể được sử dụng như một môđun Python hoặc như một giao diện dòng lệnh. Nó cung cấp các tùy chọn để chỉ định độ dài dòng tối đa, kiểu đặt tên biến, thực lề, v.v. YAPF có thể được sử dụng cùng với các công cụ phân tích mã Python khác như Flake8 và PyLint để đảm bảo chất lượng mã nhất quán.

Để đảm bảo tổ chức nhập đúng cách, chúng tôi đã sử dụng Isort, một thư viện Python sắp xếp và định dạng các mục nhập trong mã Python. Isort giúp duy trì thứ tự nhất quán và hợp lý cho các mục nhập, cải thiện khả năng tổ chức và khả năng đọc của mã. Cuối cùng, chúng tôi đã kết hợp Ruff, một thư viện Python để tái cấu trúc mã, nhằm tăng cường khả năng đọc và khả năng bảo trì của mã được tạo. Ruff tự động thực hiện các chuyển đổi mã, chẳng hạn như đơn giản hóa các biểu thức phức tạp, trích xuất các hàm và áp dụng một tái cấu trúc khác để cải thiện cấu trúc và độ rõ ràng của mã.

²<https://flake8.pycqa.org/en/latest/>

- ³<https://github.com/hhatto/autopep8>
⁴<https://github.com/asottile/add-trailing-comma>
⁵<https://github.com/google/yapf>
⁶<https://github.com/psf/black>

Bằng cách kết hợp các kỹ thuật sửa lỗi và cải tiến mã này, mã được tạo ra từ mô tả của con người có thể được tinh chỉnh, đảm bảo đáp ứng các tiêu chuẩn mã hóa Python và các thông lệ tốt nhất. Cách tiếp cận này góp phần vào độ tin cậy, khả năng bảo trì và khả năng đọc được của cơ sở mã, tạo điều kiện cho quá trình phát triển phần mềm hiệu quả. Toàn bộ quy trình hoàn chỉnh để tạo mã, sắp xếp, phân tích và sửa lỗi được thể hiện trong Hình.4.25

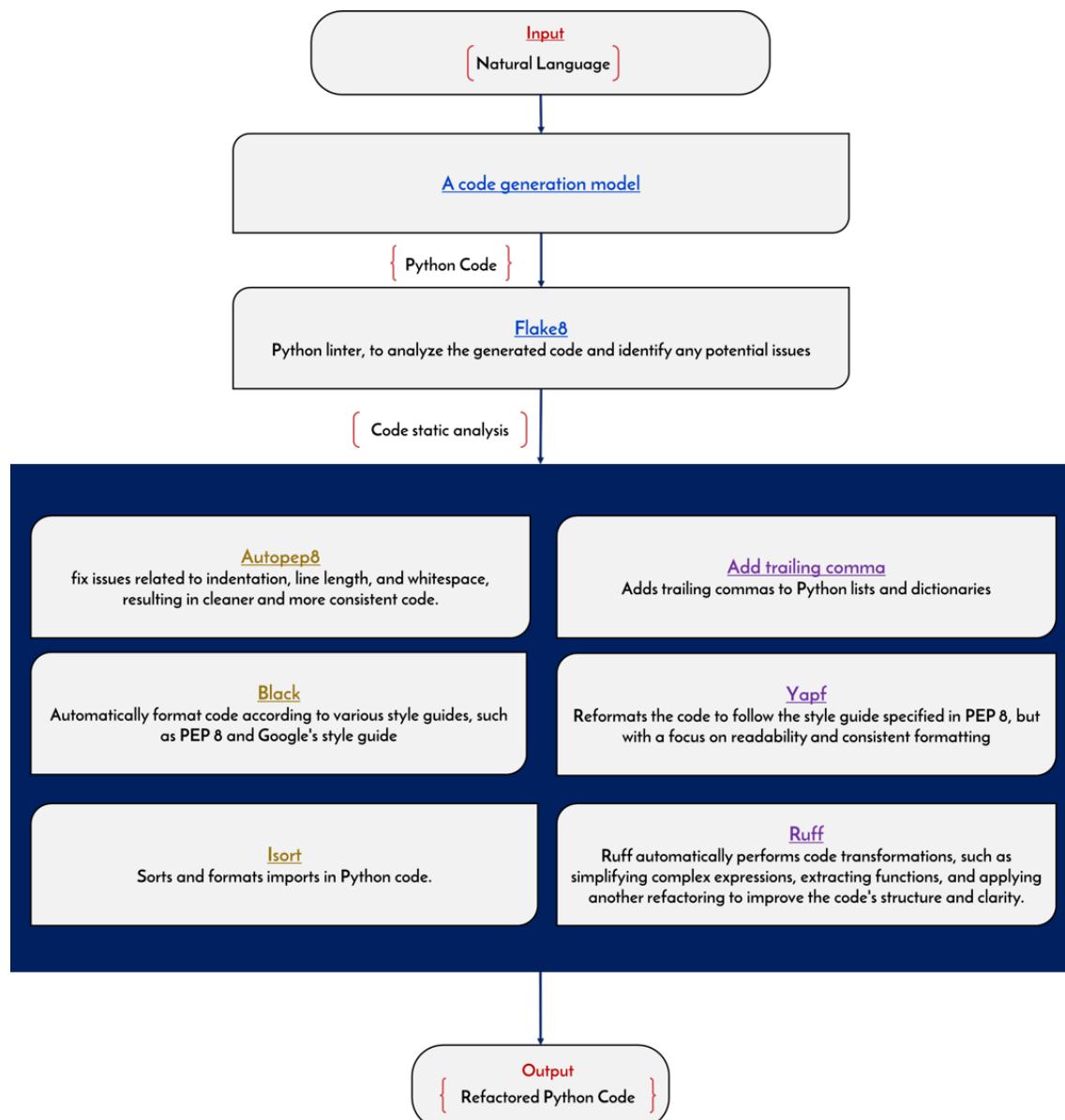


Figure 4.25: AI Assistant Code Process

Vì vậy, có ba giai đoạn để tạo mã và mã trợ lý AI như sau:

1. Tạo mã từ ngôn ngữ tự nhiên

2. 2. Kiểm tra lỗi và phân tích lỗi bằng Flake8
3. 3. Tái cấu trúc và sửa lỗi bằng các công cụ còn lại.

1.5 Summary on the Proposed Approaches

Sự xuất hiện của thế hệ mã đã trở thành một yếu tố quan trọng trong việc đơn giản hóa quy trình mã hóa cho cả các nhà phát triển dày dạn kinh nghiệm và những người mới học kỹ năng mã hóa. Chương này tập trung vào nhiệm vụ tuần tự-sang-tuần tự là chuyển đổi các giải thích ngôn ngữ tự nhiên thành các chuỗi mã thông qua các mô hình mã hóa-giải mã. Thông qua việc sử dụng các phương pháp học sâu và đào tạo trước, chúng tôi đã thiết kế hiệu quả các cấu trúc hợp lý mang lại kết quả chính xác trong khi chỉ cần tài nguyên bộ nhớ tối thiểu.

Từ công trình của mình, chúng tôi đã xác nhận và phát hiện ra những đóng góp:

1. Áp dụng và đánh giá mô hình dịch máy để hoạt động và tương tác như một mô hình giải quyết cho tác vụ tạo mã. Điều này xác nhận rằng mô hình dịch máy có thể hoạt động như một mô hình tạo mã.
2. Tận dụng các mô hình ngôn ngữ được đào tạo trước cho tác vụ tạo mã.
3. Đề xuất các mô hình lai để giải quyết vấn đề tạo mã với kích thước nhỏ trong bộ mã hóa và giải mã với kết quả dịch tuyệt vời.
4. Đảm bảo tuân thủ mã và tinh chỉnh mã với các tiêu chuẩn mã hóa và các thông lệ tốt nhất để nêu bật bằng cách sử dụng các kỹ thuật để phân tích lỗi, sửa cú pháp và tăng cường độ tin cậy và khả năng bảo trì của mã đã tạo.

Trong chương này, chúng tôi sẽ tiến hành khám phá sâu hơn về năm phương pháp tiếp cận được đề xuất để tạo mã tự động. Mỗi phương pháp tiếp cận sử dụng một mô hình mã hóa-giải mã riêng biệt, chứng minh tính linh hoạt của việc học chuyển giao và tác động của việc mã hóa chuyên biệt đối với các tác vụ tổng hợp mã. Chúng tôi đi sâu vào sự phức tạp của từng phương pháp tiếp cận, làm nổi bật các đặc điểm của việc học chuyển giao và vai trò của các bộ mã hóa trong việc nâng cao hiệu suất tạo mã.

1. Mô hình MarianCG

Phương pháp tiếp cận MarianCG tập trung vào Marian Encoder, nhấn mạnh vào các khả năng chuyên biệt của nó trong các tác vụ dịch. Đầu vào được mã hóa được xử lý bởi bộ mã hóa Marian, được thiết kế riêng cho

dịch thuật ngũ liệu song song.

Học chuyển giao diễn ra khi mô hình mã hóa-giải mã học cách dịch các chuỗi văn bản, phù hợp với bản chất giống như bản dịch cơ bản của việc tạo mã. Phương pháp tiếp cận MarianCG giới thiệu cách mã hóa cụ thể cho từng tác vụ và học chuyển giao góp phần vào việc dịch mã chính xác.

2. Mô hình RoBERTaMarian

Trong phương pháp tiếp cận RoBERTaMarian, chúng tôi khai thác sức mạnh của kiến trúc tinh chế của DistilRoBERTa. Phân tích cú pháp dựa trên từ phụ, được hỗ trợ bởi bộ phân tích cú pháp RoBERTaMarian, cho phép mô hình hiểu các cấu trúc văn bản đa dạng.

Học chuyển tiếp diễn ra khi mô hình được đào tạo trước thích ứng với các tác vụ tạo mã. Phương pháp này minh họa cho sự hiệp lực giữa phân tích cú pháp từ phụ và học chuyển tiếp, vì mô hình nắm bắt ngũ nghĩa mã trong khi vẫn duy trì hiệu quả.

3. Mô hình BERTMarian

Phương pháp BERTMarian mở rộng lợi ích của các mô hình đã tinh chế sang các tác vụ tạo mã. Với bộ mã hóa DistilBERT, học chuyển tiếp cho phép mô hình hiểu ngũ cảnh hai chiều. Bộ phân tích cú pháp BERTMarian, sử dụng phân tích cú pháp dựa trên từ phụ, đóng vai trò quan trọng trong việc phân tích các đầu vào mã phức tạp.

Phương pháp này nhấn mạnh cách học chuyển tiếp, kết hợp với phân tích cú pháp từ phụ, góp phần vào sự hiểu biết toàn diện về các cấu trúc và mối quan hệ của mã.

4. Mô hình ELECTRAMarian

Phương pháp ELECTRAMarian giới thiệu một phương pháp đào tạo trước độc đáo với sự thay thế cấp độ mã thông báo. Phương pháp này tận dụng kiến trúc và chiến lược phân tích cú pháp đặc biệt của ELECTRA. Thay thế cấp mã thông báo, được hỗ trợ bởi bộ mã thông báo ELECTRAMARIAN, tăng cường khả năng học chuyển giao bằng cách tập trung vào các mối quan hệ mã thông báo chi tiết.

Mô hình học cách dự đoán các mã thông báo được thay thế, phù hợp với các yêu cầu về độ chính xác cấp mã thông báo của thế hệ mã.

5. Mô hình LUKEMarian

Trong phương pháp LUKEMarian, các nhúng dựa trên kiến thức của LUKE

làm phong phú thêm quá trình học chuyển giao. Bộ mã thông báo LUKE Marian hỗ trợ tích hợp kiến thức chuyên ngành, phù hợp với các khả năng độc đáo của LUKE.

Học chuyển giao đạt được chiều sâu khi mô hình học từ thông tin bên ngoài, góp phần vào các tác vụ tạo mã đòi hỏi nhận thức về ngữ cảnh và kiến thức chuyên ngành.

4.5.1 Tokenization and Transfer Learning Insights

Trong suốt các cách tiếp cận này, các phương pháp tokenizer và transfer learning đóng vai trò không thể thiếu trong việc định hình kết quả tạo mã. Tokenization theo nhiệm vụ cụ thể tinh chỉnh các biểu diễn đầu vào, đảm bảo rằng các mô hình nắm bắt được các đặc điểm cụ thể của miền.

Transfer learning thu hẹp khoảng cách giữa sự hiểu biết ngôn ngữ được đào tạo trước và tổng hợp mã, cho phép các mô hình thích ứng với những thách thức độc đáo của các nhiệm vụ tạo mã.

4.5.2 Comparative Analysis

Khi so sánh các cách tiếp cận này, chúng ta thấy có sự đánh đổi giữa tính chuyên môn hóa và tính linh hoạt. Trong khi các mô hình dành riêng cho nhiệm vụ nổi trội trong các miền được chỉ định của chúng, các mô hình tinh chế ưu tiên hiệu quả. Cách tiếp cận ELECTRAMarian tập trung vào độ chính xác ở cấp độ mã thông báo, trong khi cách tiếp cận LUKE Marian tận dụng kiến thức bên ngoài. Việc lựa chọn cách tiếp cận phụ thuộc vào các yêu cầu của nhiệm vụ, vì mỗi cách tiếp cận đều kết hợp độc đáo các chiến lược mã hóa và các mô hình học chuyển giao. Bảng 4.2 cho thấy sự so sánh giữa các mô hình mã hóa được đề xuất.

Khi lĩnh vực tạo mã tiến triển, năm cách tiếp cận này mở đường cho các chiến lược kết hợp khai thác thế mạnh của nhiều mô hình. Ngoài ra, nghiên cứu sâu hơn có thể khám phá sự tích hợp của các kỹ thuật mã hóa đang phát triển và kiến trúc mã hóa-giải mã. Sự đổi mới liên tục sẽ thúc đẩy sự phát triển của quá trình tạo mã, đưa chúng ta đến với các giải pháp hiệu quả hơn, chính xác hơn và có nhận thức về miền hơn.

Tóm lại, chương này làm sáng tỏ cách mã hóa và học chuyển giao thúc đẩy các cách tiếp cận được đề xuất một cách hiệp lực, tạo ra các mô hình vượt trội trong việc tạo mã. Các cách tiếp cận này nhấn mạnh tiềm năng chuyển đổi của mã hóa được tùy chỉnh và thể hiện khả năng thích ứng của học chuyển giao trong bối cảnh tổng hợp mã tự động.

Cuối cùng, chúng ta có thể nói rằng các mô hình ngôn ngữ được đào tạo trước đã đạt được thành công đáng kinh ngạc trong Xử lý dựa trên máy biến áp, dẫn đến sự thay đổi mô hình từ học có giám sát sang đào tạo trước tiếp theo là tinh chỉnh[203]. Các ứng dụng Xử lý dựa trên máy biến áp đã chứng kiến sự gia tăng quan tâm nghiên cứu trong việc cải thiện các mô hình được đào tạo trước.

Table 4.2: Comparison of Proposed Encoder Models

Pre-trained Model	Company/University	Training Details	Data Source	Tokenizer	Representation	Functionality	Parameters
DistilBERT	HuggingFace	Reduced layers, smaller encoder and decoder	BookCorpus + English Wikipedia	WordPiece subword segmentation	Tokens	Next Sentence Prediction, Masked Language Modeling	66 Million Parameters
DistilRoBERTa	Meta (Facebook)	Modified BERT model, longer training	OpenWebTextCorpus (4x RoBERTa)	Byte-level BPE tokenizer	Tokens	Masked Language Modeling (MLM)	82 Million Parameters
ELECTRA	Stanford University	Masked entity prediction	SQuAD 2.0 dataset	WordPiece	Tokens	Efficient across all input tokens	14 Million Parameters
LUKE	Various Institutions	Entity prediction, masked words	Entity-annotated data from Wikipedia	Byte-level BPE tokenizer	Word + Entity tokens	Masked Language Model (MLM)	253 Million Parameters

Chapter 5

Experimental Results

1.1 Datasets

Bằng cách tinh chỉnh mô hình biến áp MarianMT trên các tập dữ liệu CoNaLa và DJANGO, chúng tôi đã có được MarianCG, một mô hình biến áp mới được xây dựng trên biến áp được đào tạo trước. Chúng tôi đã làm theo [178] [183] trong tập dữ liệu CoNaLa và chọn các mẫu được khai thác hàng đầu tùy thuộc vào xác suất cặp NL-Code là chính xác. Tập dữ liệu CoNaLa được tạo ra có khoảng 13K NL-Code khác nhau.

Điều này nhằm đảm bảo so sánh công bằng rằng nếu chúng tôi muốn tham gia thử thách CoNaLa, thì việc đào tạo mô hình được thực hiện bằng cách sử dụng các tập dữ liệu conala-train và/hoặc conala-mined, sau đó lấy trường ý định được viết lại từ tập dữ liệu conala-test làm đầu vào và tạo đầu ra từ đó.

Tập dữ liệu CoNaLa như đã đề cập chứa khoảng 13K NL-Code khác nhau, sau đó chúng tôi sử dụng nhiều ví dụ hơn cho một thử nghiệm khác. Chúng tôi sao chép tập dữ liệu và sử dụng 26K ví dụ NL-Code. Bộ dữ liệu này chứa conala-train và các ví dụ từ conala-mined và 500 ví dụ trong conala-test để so sánh theo cùng chuẩn mực như những người đóng góp tiên tiến khác. Ngoài ra, chúng tôi đã sử dụng DJANGO chứa 19K ví dụ và có được kết quả.

Bảng 5.1 hiển thị các bộ dữ liệu được sử dụng trong mỗi thử nghiệm, cũng như kích thước bộ dữ liệu và số lượng bản ghi trong các bộ dữ liệu đào tạo, xác thực và thử nghiệm.

Table 5.1: Datasets in each experiment and distribution of the data

Dataset	Dataset Size	Dataset split		
		Train	Validation	Test
CoNaLa	13K	11125	1237	500
DJANGO	19K	16000	1000	1805
CoNaLa Large	26K	24687	1237	500

Bộ dữ liệu chứa khoảng 13K NL-Code khác nhau như thể hiện trong Hình 5.1. Bộ dữ liệu này chứa conala-train và các ví dụ từ conala-mined và 500 ví dụ trong conala-test để so sánh theo cùng chuẩn mực như những người đóng góp tiên tiến khác. Ngoài ra, chúng tôi triển khai MarianCG để điều chỉnh bộ dữ liệu DJANGO có khoảng 19K cặp NL-Code như thể hiện trong Hình 5.2.



Figure 5.1: CoNaLa Dataset(Training-Validation-Testing)



Figure 5.2: DJANGO Dataset(Training-Validation-Testing)

Chúng tôi nhận thấy kết quả cao và chính xác trong quá trình đào tạo và thử nghiệm DJANGO. Vì vậy, chúng tôi quyết định thực hiện một quy trình đào tạo khác trên tập dữ liệu CoNaLa với nhiều dữ liệu hơn và giữ nguyên kiến trúc mô hình MarianMT và tinh chỉnh mô hình được đào tạo trước này trên tập dữ liệu CoNaLa 26K NL-Code như thể hiện trong Hình 5.3.

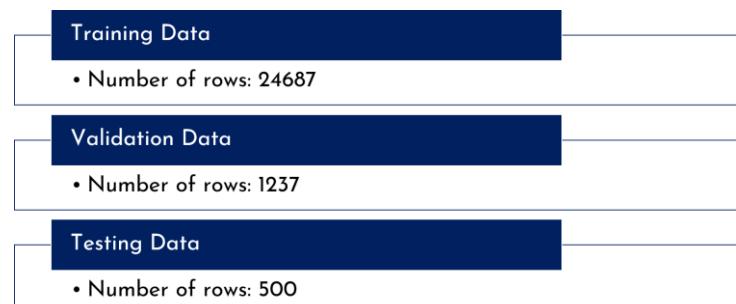


Figure 5.3: CoNaLa Dataset(Training-Validation-Testing)

1.2 Experimental setup

Trong các thí nghiệm của mình, chúng tôi đã sử dụng DJANGO và tập dữ liệu CoNaLa gồm 26K cặp NL-Code từ CoNaLa-train và CoNaLa-mined. Tập dữ liệu thử nghiệm chứa 500 mẫu từ conala-test được so sánh bằng cùng chuẩn mực như những người đóng góp tiên tiến khác.

Việc triển khai các mô hình được tạo ra của chúng tôi được thực hiện bằng Google Colab Pro. Quá trình phát triển của chúng tôi được thực hiện trên Python và PyTorch, và nhờ các mô hình biến đổi được đào tạo trước của HuggingFace, trong đó chúng tôi sử dụng năm mô hình ngôn ngữ được đào tạo trước từ trung tâm mô hình của họ. Đối với quá trình đào tạo, chúng tôi dựa vào trình đào tạo HuggingFace và việc triển khai trình lập lịch tốc độ học với kích thước lô bằng 2.

Chúng tôi đã sử dụng các tham số sau trong quá trình triển khai:

- Adam[204] as our optimizer
- Learning rate = $1e^{-5}$
- Weight decay=0.01
- Maximum length = 512
- Maximum position embeddings = 512
- Number of hidden layers = 6
- scale embedding = true
- Activation function = swish
- A linear learning rate scheduler
- A warmup ratio of 0.05
- For generation, we used beam search with four beams, early stopping, and a length penalty of 0.9.

1.3 Experimental Results

Trong phần này, chúng tôi trình bày kết quả thử nghiệm của chúng tôi về tác vụ tạo mã bằng mô hình MarianCG và mô hình lai được đề xuất của chúng tôi, được đánh giá trên các tập dữ liệu CoNaLa và DJANGO. Đối với các thử nghiệm MarianCG, chúng tôi đã đo hiệu suất tạo mã của mô hình bằng cách tính toán các số liệu như độ chính xác của mã và điểm tương đồng của mã. Kết quả cho thấy MarianCG đạt được độ chính xác tạo mã cạnh tranh trên cả hai tập dữ liệu, chứng minh tính hiệu quả của nó trong việc tạo các đoạn mã cú pháp chính xác.

Chúng tôi đã giới thiệu mô hình lai mới của mình, tận dụng thế mạnh của MarianCG trong khi kết hợp thông tin ngữ cảnh bổ sung từ nhúng BERT và nhiều mô hình chỉ dành cho bộ mã hóa. Mô hình lai đã cải thiện đáng kể chất lượng mã được tạo ra, đạt được độ chính xác của mã cao hơn và tính liên quan về mặt ngữ nghĩa so với MarianCG trên cả hai tập dữ liệu. Những phát hiện này chứng minh tính hiệu quả của phương pháp lai được đề xuất của chúng tôi trong việc nâng cao hiệu suất tạo mã, biến nó thành giải pháp đầy hứa hẹn cho các tác vụ Xử lý dựa trên bộ biến đổi liên quan đến mã.

1.3.1 MarianCG Model

1.3.1.1 Experiment 1

Chúng tôi đã đào tạo mô hình MarianCG bằng cách sử dụng chuẩn dữ liệu DJANGO, trong đó có các ví dụ về Ngôn ngữ tự nhiên làm đầu vào và Mã làm đầu ra. Tập dữ liệu DJANGO chứa 19K mục nhập ghép nối ngôn ngữ tự nhiên và mã. Tập dữ liệu này là một trong những tập dữ liệu được sử dụng phổ biến nhất cho công việc này và việc triển khai và đào tạo mô hình MarianCG của chúng tôi đã đưa ra các dự đoán có độ chính xác cao.

Bảng 5.2 hiển thị kết quả của các mô hình tạo mã trên tập dữ liệu DJANGO. MarianCG được coi là mô hình tuyệt vời nhất trong thử thách DJANGO. MarianCG có điểm BLEU là 90,41 và ghi lại độ chính xác khớp chính xác là 81,83%. So sánh tất cả các giá trị được hiển thị trong Hình 5.4. Mô hình MarianCG được đào tạo trên tập dữ liệu DJANGO có sẵn thông qua Hugging Face hub¹.

Table 5.2: Results of MarianCG model on DJANGO dataset

Rank	Model	BLEU Score	Exact Match Accuracy	Year
1	MarianCG (Ours) [2]	90.41	81.83	2022
2	TranX + BERT w/ mined [184]	79.86	81.03	2022
3	BERT + TAE [182]	-	81.77	2021
4	Reranker [176]	-	80.2	2019
5	TranX [175]	-	73.7	2018
6	ipn [5]	77.6	62.3	2016
7	Phrasal Statistical MT [5]	47.6	31.5	2016

1.3.1.2 Experiment 2

Thí nghiệm này cho thấy việc tinh chỉnh và tạo ra mô hình biến áp

MarianCG bằng cách sử dụng tập dữ liệu CoNaLa. Nó có hai phần về đào tạo với số lượng

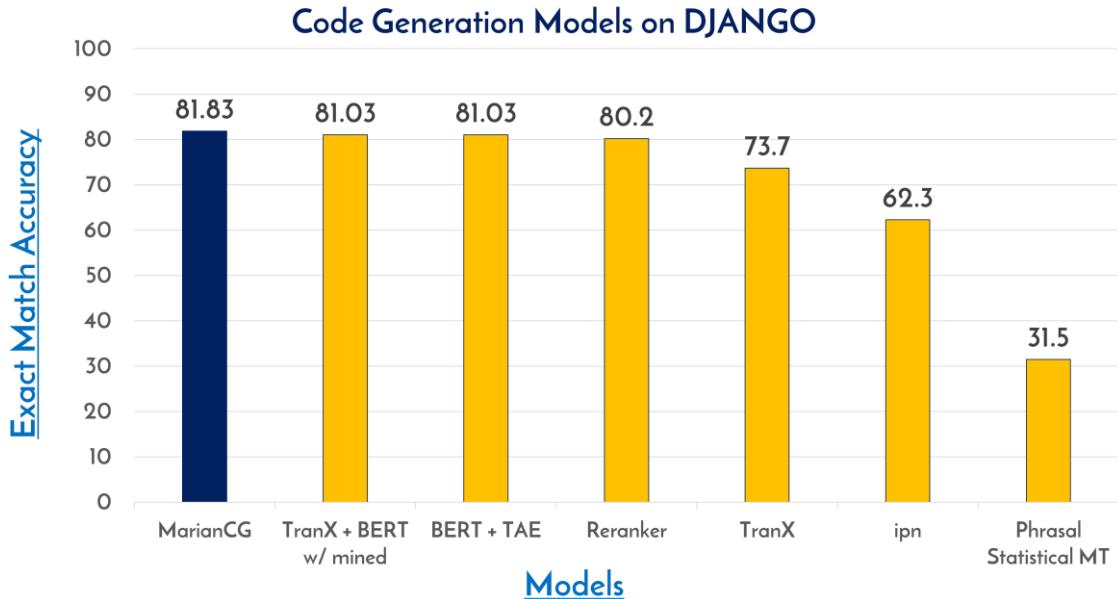


Figure 5.4: Results on DJANGO

ghi chép trên tập dữ liệu CoNaLa. Đầu tiên, thí nghiệm tạo ra một tập dữ liệu gồm khoảng 13K cặp ngôn ngữ tự nhiên và mã.

Bảng 5.3 cho thấy kết quả của thí nghiệm trên tập dữ liệu CoNaLa, trong đó các dự đoán của mô hình MarianCG đã thúc đẩy mô hình này trở thành một trong những mô hình tạo mã chính xác hàng đầu về độ chính xác và điểm BLEU.

Table 5.3: Results of the first experiment on CoNaLa

Model	BLEU Score	Exact Match Accuracy	Year
TranX + BERT w/ mined [184]	34.2	5.8	2022
BERT + TAE [182]	33.41	-	2021
External Knowledge With API + Reranking [178]	32.26	-	2020
MarianCG (Ours) [2]	30.92	6.2	2022
External Knowledge With API [178]	30.69	-	2020
BART W/ Mined [183]	30.55	-	2021
Reranker [176]	30.11	2.8	2019
BART Base [183]	26.24	-	2021
TranX [175]	24.3	-	2018

Trong trường hợp này, mô hình MarianCG đạt được độ chính xác khớp chính xác cao nhất trong tất cả các mô hình. Thí nghiệm này đạt được điểm BLEU là 30,92 và đạt được 6,2% độ chính xác khớp chính xác với lợi thế là sự chú

ý của nhiều đầu và dễ sử dụng thông qua Hugging Face hub².

The following are some examples of employing MarianCG for the code generation task on CoNaLa.

Input: check if all elements in list ‘mylist’ are identical

Ref:

```
all(x == myList[0] for x in myList)
```

Output:

```
all(isinstance(x, int) for x in mylist)
```

Bleu score: 19.34

Input: find intersection data between series ‘s1’ and series ‘s2’

Ref:

```
pd.Series(list(set(s1).intersection(set(s2))))
```

Output:

```
s1.intersection(s1, s2.intersection(s2))
```

Bleu score: 24.55

Input: get the first object from a queryset in django model ‘entry’

Ref:

```
Entry.objects.filter()[:1].get()
```

Output:

```
Entry.objects.filter(id=1).order_by('id')[0]
```

Bleu score: 30.12

Chúng tôi phát hiện ra rằng việc thu thập thêm dữ liệu dẫn đến kết quả chính xác hơn. Vì vậy, đối với thí nghiệm tiếp theo, chúng tôi đã đào tạo mô hình MarianCG với số lượng mẫu đào tạo tăng lên. Vì vậy, trong thí nghiệm này, chúng tôi đã sử dụng lại tập dữ liệu CoNaLa, nhưng lần này với 26K bản ghi.

Điều này đáp ứng được kỳ vọng của chúng tôi và đưa mô hình MarianCG lên vị trí hàng đầu trong thử thách CoNaLa. Kết quả dữ liệu thử nghiệm mới không giống với thử nghiệm ban đầu của chúng tôi. Thí nghiệm này mang lại Điểm BLEU là 34,43 và độ chính xác khớp chính xác là 10,2%. So sánh mô hình của chúng tôi với các mô hình chuẩn CoNaLa sau thí nghiệm này cho thấy mô hình MarianCG có dự đoán chính xác nhất khi so sánh với các mô hình tiên tiến khác. Điều này được hiển thị trong bảng 5.4 so sánh kết quả của thí nghiệm này với các mô hình khác trong thử thách tạo mã CoNaLa, hiển thị Điểm BLEU và độ chính xác khớp chính xác của từng mô hình.

Table 5.4: Results of MarianCG model on the CoNaLa dataset

Rank	Model	BLEU Score	Exact Match Accuracy	Year
1	MarianCG (Ours) [2]	34.43	10.2	2022
2	TranX + BERT w/ mined [184]	34.2	5.8	2022
3	BERT + TAE [182]	33.41	-	2021
4	External Knowledge With API + Reranking [178]	32.26	-	2020
5	External Knowledge With API [178]	30.69	-	2020
6	BART W/ Mined [183]	30.55	-	2021
7	Reranker [176]	30.11	2.8	2019
8	BART Base [183]	26.24	-	2021
9	TranX [175]	24.3	-	2018

Ngoài ra, Hình 5.5 mô tả tất cả các kết quả của tập dữ liệu CoNaLa. Ngoài ra, mô hình MarianCG có sẵn thông qua trung tâm Hugging Face3.

Từ các thí nghiệm trên mô hình MarianCG [2], chúng tôi nhận thấy rằng mô hình này đạt điểm BLEU 34,43, SacreBLEU là 30,68, điểm ROUGE của kết quả là 49,63 và độ chính xác khớp chính xác là 10,2% trên tập dữ liệu CoNaLa. Ngoài ra, các kết quả trên tập dữ liệu DJANGO đạt điểm BLEU 90,1 và độ chính xác khớp chính xác là 81,83%.

Mô hình MarianCG được xếp hạng trong số các mô hình cao nhất về khả năng dự đoán chính xác về điểm BLEU và độ chính xác khớp chính xác. Mô hình này có kiến trúc kích thước nhỏ hơn. Nó có sáu lớp trong bộ mã hóa và sáu lớp trong bộ giải mã, trong khi các mô hình khác có kích thước mô hình lớn hơn.

³<https://huggingface.co/AhmedSSoliman/MarianCG-CoNaLa-Large>

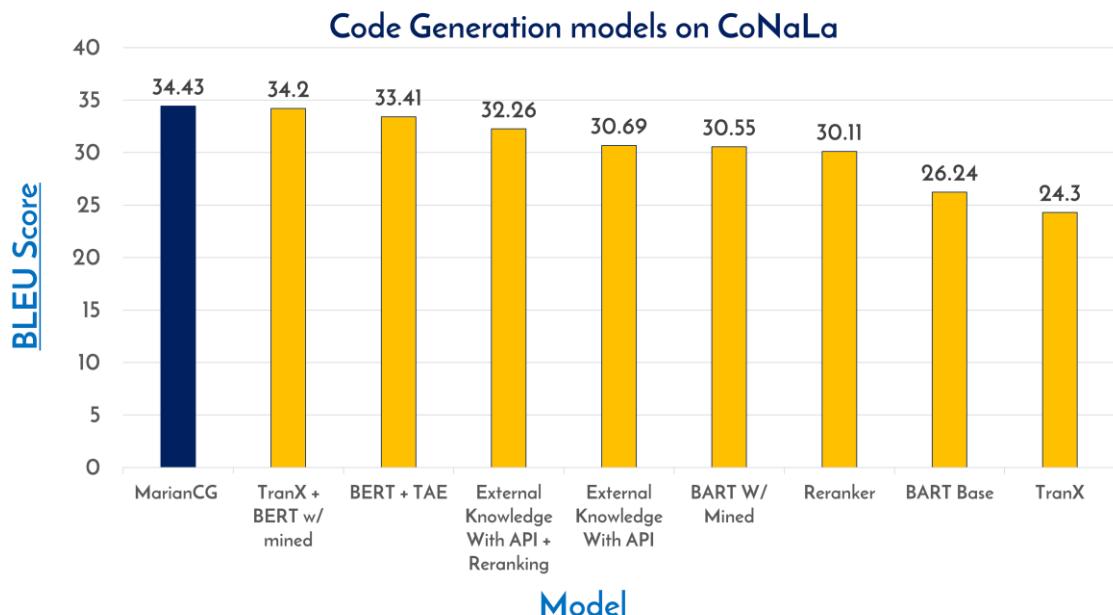


Figure 5.5: Results on CoNaLa

1.3.2 RoBERTaMarian Model

Khi được đánh giá trên tập dữ liệu CoNaLa, mô hình RoBERTaMarian cho thấy hiệu quả của nó bằng cách đạt được các số liệu hiệu suất ấn tượng, bao gồm điểm BLEU là 35,74, độ chính xác khớp chính xác là 13,8%, Sacrebleu là 31,30 và điểm ROUGE là

44,25. Điều này chứng minh cho hiệu quả của mô hình và khả năng tạo ra các giải pháp mã chất lượng cao. Hơn nữa, khi được đánh giá trên tập dữ liệu DJANGO, mô hình RoBERTaMarian chứng minh được tính vượt trội của nó so với tất cả các mô hình tiên tiến trong việc tạo mã, tự hào có điểm BLEU vượt trội là 88,91, độ chính xác khớp chính xác là 77,95%, Sacrebleu là 74,08 và điểm ROUGE là 92,76.

Những kết quả nổi bật này càng xác nhận thêm khả năng đặc biệt của mô hình RoBERTaMarian, củng cố vị thế của nó như một giải pháp tiên tiến cho các tác vụ tạo mã. Mô hình RoBERTaMarian thể hiện một số lợi thế chính khiến nó trở thành giải pháp vượt trội cho các tác vụ tạo mã.

Mô hình RoBERTaMarian cung cấp một loạt các lợi thế chính khiến nó trở thành giải pháp thực sự đặc biệt cho các tác vụ tạo mã. Một trong những điểm mạnh đáng chú ý nhất của nó là độ chính xác được cải thiện mà nó mang lại. Bằng cách kết hợp bộ giải mã Marian với mô hình được đào tạo trước DistilRoBERTa mạnh mẽ làm bộ mã hóa, mô hình RoBERTaMarian tạo ra các giải pháp mã với độ chính xác và tinh tế đáng kinh ngạc. Các đầu ra kết quả phù hợp chặt chẽ với các

kết quả mong muốn, giảm lỗi và đảm bảo tạo mã chất lượng cao. Hơn nữa, mô hình này chứng minh độ tin cậy và tính nhất quán không lay chuyển trên các tập dữ liệu và số liệu đánh giá khác nhau. Bất kể bối cảnh lập trình nào, mô hình RoBERTaMarian luôn cung cấp các đầu ra mã đáng tin cậy. Khía cạnh này nhấn mạnh sự đáng tin cậy của nó như một công cụ đáng tin cậy để tạo mã, bất kể mức độ phức tạp của nhiệm vụ đang thực hiện.

Một tính năng nổi bật khác là điểm BLEU ấn tượng của mô hình, thiết lập chuẩn mực mới trong việc tạo mã. Với điểm BLEU là 35,74 trên tập dữ liệu CoNaLa và điểm nổi bật là 88,91 trên tập dữ liệu DJANGO, mô hình RoBERTaMarian thể hiện khả năng đáng chú ý trong việc tạo ra các giải pháp mã phù hợp chặt chẽ với các tham chiếu do con người tạo ra. Điểm BLEU cao này làm nổi bật khả năng của mô hình trong việc nắm bắt cả tính chính xác về mặt ngữ nghĩa và cú pháp của mã được tạo ra, thúc đẩy tiện ích và độ tin cậy của mô hình trong các ứng dụng thực tế.

Mô hình RoBERTaMarian cũng vượt trội về độ chính xác khớp chính xác, một số liệu quan trọng trong các tác vụ tạo mã. Với độ chính xác là 13,8% trên CoNaLa và 77,95% đáng chú ý trên DJANGO, mô hình luôn tạo ra các giải pháp mã khớp chính xác với đầu ra mong muốn. Độ chính xác đặc biệt này rất cần thiết để đảm bảo tính chính xác và độ tin cậy của mã được tạo ra, biến mô hình thành một tài sản có giá trị trong các tình huống lập trình thực tế. Không thể bỏ qua điểm Sacrebleu đáng chú ý của mô hình. Với số điểm 31,30 trên CoNaLa và 74,08 trên DJANGO, mô hình RoBERTaMarian thể hiện sức mạnh ngôn ngữ của mình, đánh giá hiệu quả tính trôi chảy và tính đầy đủ của mã do máy tạo ra. Tính năng này cung cấp thêm khả năng của mô hình trong việc tạo ra các giải pháp mã giống con người, một khía cạnh quan trọng của việc tự động hóa các tác vụ lập trình.

Ngoài các số liệu ấn tượng này, mô hình RoBERTaMarian còn tỏa sáng với số điểm ROUGE nổi bật là 44,25 trên CoNaLa và 92,76 trên DJANGO. Khả năng tạo ra các giải pháp mã phù hợp với các tham chiếu chuẩn vàng của mô hình chứng minh khả năng nắm bắt ngữ nghĩa mã quan trọng một cách hiệu quả.

Hơn nữa, vượt trội hơn các mô hình tiên tiến khác trong các tác vụ tạo mã, mô hình RoBERTaMarian là giải pháp tiên tiến trong lĩnh vực này. Hiệu suất đáng chú ý của mô hình trên các tập dữ liệu đa dạng cung cấp vị thế của mô hình như một đối thủ đáng gờm, mở rộng ranh giới của công nghệ tạo mã và thúc đẩy công nghệ tiên tiến.

Ngoài các số liệu hiệu suất, tính linh hoạt và khả năng khai thác hóa của mô hình còn thể hiện rõ qua thành công của nó trên nhiều tập dữ liệu và bối cảnh lập trình. Với khả năng thích ứng với nhiều tình huống tạo mã khác nhau, mô hình RoBERTaMarian chứng minh được tiềm năng ứng dụng rộng rãi, hỗ trợ nhiều ngôn ngữ lập trình và miền văn đề khác nhau.

Cuối cùng, độ chính xác, độ tin cậy, độ đúng đắn và hiệu suất tổng thể ấn tượng của mô hình RoBERTaMarian hứa hẹn rất nhiều cho các ứng dụng thực tế trong quá trình tạo mã tự động. Tiềm năng cải thiện đáng kể quá trình phát triển phần mềm và tự động hóa khiến nó trở thành một tài sản mạnh mẽ trong

lĩnh vực lập trình và tiến bộ công nghệ.

The following are examples of employing the RoBERTaMarian model for the code generation task.

Input: check if all elements in list ‘mylist’ are identical

Ref:

```
all(x == myList[0] for x in myList)
```

Output:

```
all(x == myList[0] for x in myList)
```

Bleu score: 100.0

Input: create 3d array of zeroes of size ‘(3,3,3)’

Ref:

```
numpy.zeros((3, 3, 3))
```

Output:

```
np.zeros((3, 3, 3))
```

Bleu score: 91.93

1.3.3 BERTMarian Model

Mô hình BERTMarian là mô hình thứ ba bao gồm bộ mã hóa DistilBERT và Bộ giải mã Marian. Để đánh giá hiệu suất của nó, chúng tôi đã tiến hành đánh giá trên tập dữ liệu CoNaLa và kết quả chứng minh các tính năng chính sau. Trên tập dữ liệu CoNaLa, mô hình đạt điểm BLEU là 32,46, cho thấy khả năng tạo ra các giải pháp mã khớp chặt chẽ với các tham chiếu do con người tạo ra.

Hơn nữa, độ chính xác khớp chính xác là 12,4% cho thấy khả năng tạo ra các đầu ra mã khớp chính xác với các giải pháp mong muốn của mô hình, nâng cao độ tin cậy và tính chính xác của mã do nó tạo ra. Về mặt đánh giá ngôn ngữ, điểm SacreBLEU là 29,48 phản ánh sự trôi chảy và tính đầy đủ của mô hình trong việc tạo ra các giải pháp mã. Điểm SacreBLEU cao hơn cho thấy sự tương đồng về ngôn ngữ lớn hơn giữa mã được tạo ra và mã do con người tạo ra.

Ngoài ra, điểm ROUGE là 43,95 nhấn mạnh khả năng của mô hình trong việc nắm bắt ngữ nghĩa mã thiết yếu một cách hiệu quả, góp phần vào tính mạch lạc và tính liên quan của mã được tạo ra.

Sau đây là các ví dụ về việc sử dụng mô hình BERTaMarian cho nhiệm vụ tạo mã.

Input: get the date 7 days before the current date

Ref:

```
(datetime.datetime.now() - datetime.timedelta(days=7)).date()
```

Output:

```
( datetime.datetime.now() - datetime.timedelta(days = 1) )
```

Bleu score: 69.74

Input: download a file ‘url’ over http

Ref:

```
r = requests.get(url)
```

Output:

```
response = requests.get(url)
```

Bleu score: 88.01

Mở rộng đánh giá của chúng tôi sang tập dữ liệu DJANGO, mô hình thứ ba duy trì hiệu suất mạnh mẽ của nó với điểm BLEU là 56,55, xác nhận thêm khả năng tạo ra các giải pháp mã chất lượng cao của nó. Ngoài ra, độ chính xác khớp chính xác là 76,78% đảm bảo khả năng tạo ra các đầu ra mã chính xác của mô hình, phù hợp chính xác với các mục tiêu đã chỉ định.

Luôn thể hiện sức mạnh ngôn ngữ của mình, điểm SacreBLEU là 29,48 trên tập dữ liệu DJANGO cung cấp tính trôi chảy và đầy đủ của mô hình trong việc tạo mã. Hơn nữa, điểm ROUGE ấn tượng là 43,95 làm nổi bật hiệu quả của mô hình trong việc nắm bắt ngữ nghĩa mã quan trọng, tăng cường tính mạch lạc và tính liên quan của mã được tạo ra.

Tóm lại, các tính năng chính của mô hình BERTMarian bao gồm điểm BLEU mạnh, biểu thị khả năng tạo ra các giải pháp mã gần giống với mã do con người tạo ra. Độ chính xác khớp chính xác của nó đảm bảo độ tin cậy và độ chính xác của đầu ra. Hơn nữa, điểm SacreBLEU và ROUGE đáng chú ý của mô hình nhấn mạnh trình độ ngôn ngữ và khả năng nắm bắt ngữ nghĩa mã quan trọng của nó.

1.3.4 ELECTRAMarian Model

Giới thiệu mô hình mã hóa-giải mã thứ tư của chúng tôi, mô hình ELECTRAMarian tận dụng sự kết hợp mạnh mẽ của ELECTRA làm bộ mã hóa và Marian Decoder. Kiến trúc tiên tiến này cho thấy kết quả đầy hứa hẹn trên tập dữ liệu CoNaLa, với điểm BLEU là 30,18. Độ chính xác khớp chính xác của mô hình là 10,0%, chứng minh khả năng tạo ra các giải pháp mã khớp chính xác với các đầu ra đã chỉ định, góp phần vào độ tin cậy và tính chính xác của mã được tạo ra. Sau đây là các ví dụ về việc sử dụng mô hình ELECTRAMarian cho tác vụ tạo mã.

Input: get the last part of a string before the character '-'

Ref:

```
print(x.rsplit('-', 1)[0])
```

Output:

```
print ( my _ string. split ('[ ^ - 9 ]') [ 0 ] )
```

Bleu score: 33.7

Input: write the elements of list 'lines' concatenated by special character 'n' to file 'myfile'

Ref:

```
myfile.write('\n'.join(lines))
```

Output:

```
print ('\n '. join ( lines ) )
```

Bleu score: 78.82

Sau khi đánh giá toàn diện các số liệu còn lại trên tập dữ liệu CoNaLa, phép đo SacreBLEU đạt được số điểm đáng khen là 27,15. Điểm số này phản ánh sự trôi chảy về mặt ngôn ngữ và tính đầy đủ của mô hình trong việc tạo ra các giải pháp mã, một khía cạnh quan trọng của các tác vụ tạo mã. Ngoài ra, điểm số ROUGE là 42,42 nhấn mạnh khả năng của mô hình trong việc nắm bắt hiệu quả ngữ nghĩa mã quan trọng, qua đó nâng cao hơn nữa tính mạch lạc và tính liên quan của mã được tạo ra.

Chuyển sự chú ý của chúng ta sang tập dữ liệu DJANGO, mô hình ELECTRAMarian thể hiện ấn tượng tính linh hoạt và hiệu suất cao của nó. Với số điểm BLEU đáng chú ý là 53,02, nó vượt trội trong việc tạo ra các giải pháp mã phù hợp chặt chẽ với các tài liệu tham khảo do con người biên soạn. Độ chính xác khớp chính xác, đo được ở mức ấn tượng là 65,32%, đảm bảo khả năng thành thạo của mô hình trong việc tạo ra các đầu ra mã chính xác, chứng minh tính mạnh mẽ của nó trong một kịch bản thực tế.

Về mặt đánh giá ngôn ngữ trên tập dữ liệu DJANGO, ELECTRAMarian đạt được giá trị đáng chú ý là 58,16 điểm Sacrebleu, nhấn mạnh tính lưu loát về ngôn ngữ và tính đầy đủ của mô hình trong việc tạo mã. Hơn nữa, mô hình thể hiện hiệu suất đặc biệt trong việc nắm bắt ngữ nghĩa mã quan trọng, như bằng chứng từ điểm ROUGE ấn tượng là 83,91, xác nhận thêm hiệu quả của nó trong việc tạo ra các giải pháp mã mạch lạc và có liên quan theo ngữ cảnh.

Cấu hình trong mô hình ELECTRAMarian thể hiện các khả năng đầy hứa hẹn trong việc tạo mã. Điểm BLEU cạnh tranh, độ chính xác khớp chính xác, điểm SacreBLEU và điểm ROUGE trên cả tập dữ liệu CoNaLa và DJANGO chứng minh tiềm năng của nó như một công cụ có giá trị cho nhiều tác vụ tạo mã khác nhau, khẳng định vị thế của nó như một giải pháp tiên tiến trong lĩnh vực lập trình và Xử lý dựa trên máy biến áp.

5.3.5 LUKE^Marian Model

Giới thiệu Mô hình LUKE^Marian, có sự kết hợp động giữa bộ mã hóa LUKE và Bộ giải mã Marian, tạo nên giải pháp tạo mã mạnh mẽ và hiệu quả. Sau khi đánh giá tập dữ liệu CoNaLa, mô hình (LUKE + Bộ giải mã Marian) đạt điểm BLEU đáng chú ý là 29,83, cho thấy khả năng tạo ra các giải pháp mã phù hợp chặt chẽ với các tham chiếu do con người tạo ra.

Ngoài ra, mô hình thể hiện độ chính xác khớp chính xác là 7,6%, chứng minh khả năng tạo ra các đầu ra mã khớp chính xác với các mục tiêu đã chỉ định, do đó cung cấp độ chính xác và độ tin cậy của mã được tạo ra.

Phân tích sâu hơn về các số liệu của tập dữ liệu CoNaLa cho thấy điểm SacreBLEU là

25,32 cho mô hình, làm nổi bật sự trôi chảy về mặt ngôn ngữ và tính đầy đủ của mô hình trong việc tạo ra các giải pháp mã. Hơn nữa, điểm ROUGE của mô hình đạt giá trị ấn tượng là 39,84, cho thấy hiệu quả của mô hình trong việc nắm bắt

ngữ nghĩa mã quan trọng, dẫn đến việc tạo ra các đầu ra mã mạch lạc và có liên quan theo ngữ cảnh.

Trên tập dữ liệu DJANGO, mô hình LUKEMarian tỏa sáng với các số liệu hiệu suất đáng chú ý. Với điểm BLEU nổi bật là 89,34, mô hình này xuất sắc trong việc tạo ra các giải pháp mã phù hợp chặt chẽ với các tham chiếu chuẩn vàng, xác nhận thêm hiệu quả của nó như một giải pháp tạo mã hàng đầu. Hơn nữa, độ chính xác khớp chính xác là 78,50% nhấn mạnh khả năng của mô hình trong việc tạo ra các đầu ra mã chính xác một cách nhất quán, chứng minh tính mạnh mẽ và độ tin cậy của nó trong các ứng dụng thực tế.

Sau đây là các ví dụ về việc sử dụng mô hình LUKEMarian cho tác vụ tạo mã.

Input: check if directory ‘directory’ exists and create it if necessary

Ref:

```
if (not os.path.exists(directory)):  
    os.makedirs(directory)
```

Output:

```
if (not os.path.exists(directory)):  
    os.makedirs(directory)
```

Bleu score: 100.0

Input: using %f with strftime() in python to get microseconds

Ref:

```
datetime.datetime.now().strftime('%H:%M:%S.%f')
```

Output:

```
datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')
```

Bleu score: 65.15

Về mặt đánh giá ngôn ngữ trên tập dữ liệu DJANGO, điểm Sacrebleu đạt giá trị ấn tượng là 74,66, khẳng định lại tính lưu loát và đầy đủ về mặt ngôn ngữ của mô hình trong việc tạo mã. Ngoài ra, điểm ROUGE đặc biệt của mô hình là 93,11 cho thấy khả năng nắm bắt ngữ nghĩa mã thiết yếu một cách hiệu

quả, tạo ra các giải pháp mã có tính mạch lạc cao và phù hợp với ngữ cảnh.

Mô hình LUKE Marian, có bộ mã hóa LUKE và Bộ giải mã Marian, trình bày một giải pháp tạo mã có khả năng cao và hiệu quả. Điểm BLEU cạnh tranh, độ chính xác khớp chính xác, điểm SacreBLEU và ROUGE trên cả tập dữ liệu CoNaLa và DJANGO càng nhấn mạnh thêm tiềm năng của mô hình như một công cụ vô giá cho các tác vụ tạo mã đa dạng. Khả năng tạo ra đầu ra mã chính xác, có liên quan và phù hợp với ngữ cảnh của mô hình củng cố vị thế của mô hình như một giải pháp tiên tiến đi đầu trong lĩnh vực lập trình và Xử lý dựa trên Transformers.

1.4 Discussion

Trong luận án này, chúng tôi đã khám phá năm mô hình riêng biệt cho các tác vụ tạo mã, mỗi mô hình tận dụng các bộ mã hóa được đào tạo trước khác nhau kết hợp với Marian Decoder. Các mô hình được kiểm tra là MarianCG, đại diện cho đường cơ sở tiên tiến, BERTMarian (sử dụng distilBERT làm bộ mã hóa và Marian Decoder), ELECTRAMar-ian (sử dụng ELECTRA làm bộ mã hóa và Marian Decoder), LUKE Marian (bao gồm LUKE làm bộ mã hóa và Marian Decoder) và RoBERTaMarian (kết hợp distilRoBERTa làm bộ mã hóa và Marian Decoder).

- MarianCG, là một mô hình đường cơ sở, đã thể hiện hiệu suất đáng khen ngợi với điểm BLEU là 34,43 và độ chính xác khớp chính xác là 10,2% trên chuẩn CoNaLa. Tuy nhiên, nó đã bị các mô hình bộ mã hóa-giải mã khác vượt qua trong hầu hết các số liệu hiệu suất. Tuy nhiên, trên tập dữ liệu DJANGO, MarianCG đã thể hiện được điểm mạnh của mình, đạt điểm BLEU là 90,41 và ghi nhận độ chính xác khớp chính xác đặc biệt là 81,83%.
- Mô hình RoBERTaMarian đã chứng minh hiệu suất đáng chú ý trên tập dữ liệu CoNaLa, đạt điểm BLEU là 35,74, cho thấy khả năng tạo ra các giải pháp mã khớp chặt chẽ với các tham chiếu do con người tạo ra. Độ chính xác khớp chính xác là 13,8% đã xác nhận thêm độ chính xác và độ tin cậy của mã được tạo ra. Hơn nữa, mô hình này còn xuất sắc trong việc nắm bắt ngữ nghĩa mã quan trọng, bằng chứng là điểm ROUGE là 44,25. Tương tự như vậy, trên tập dữ liệu DJANGO, RoBERTaMarian đã vượt trội hơn các mô hình khác với điểm BLEU là 88,91, chứng minh được sức mạnh của nó trên các tập dữ liệu và bối cảnh lập trình khác nhau.
- Mô hình BERTMarian đã thể hiện kết quả cạnh tranh trên tập dữ liệu CoNaLa, với điểm BLEU là 32,46 và độ chính xác khớp chính xác là 12,4%. Mặc dù điểm số thấp hơn một chút so với RoBERTaMarian, hiệu suất của mô hình vẫn đầy hứa hẹn. Trên tập dữ liệu DJANGO, BERTMarian vẫn duy trì hiệu quả của mình với điểm BLEU là 53,02 và độ chính xác khớp chính xác là 65,32%, cho thấy khả năng thích ứng và độ mạnh mẽ của nó trong một kịch bản thực tế.

- Mô hình ELECTRAMarian đã chứng minh các khả năng đáng chú ý trên tập dữ liệu CoNaLa, đạt điểm BLEU là 30,18. Mặc dù độ chính xác khớp chính xác là 10,0% tương đối thấp hơn, nhưng mô hình đã thể hiện năng lực trong

tạo ra các giải pháp mã có liên quan. Điểm SacreBLEU là 27,15 và điểm ROUGE là 42,42 đã xác nhận thêm khả năng lưu loát ngôn ngữ và khả năng nắm bắt ngữ nghĩa mã của nó. Trên tập dữ liệu DJANGO, ELECTRAMarian đã đạt được điểm BLEU đáng chú ý là 89,34, làm nổi bật khả năng tạo ra đầu ra mã chất lượng cao của nó.

- Mô hình LUKEMarian cho thấy kết quả cạnh tranh trên tập dữ liệu CoNaLa, với điểm BLEU là 29,83 và độ chính xác khớp chính xác là 7,6%. Mặc dù độ chính xác thấp hơn một chút so với các mô hình khác, nhưng sự lưu loát về ngôn ngữ đã được chứng minh với điểm SacreBLEU là 25,32 và điểm ROUGE là 39,84. Trên tập dữ liệu DJANGO, LUKEMarian nổi trội với điểm BLEU là 89,34 và độ chính xác khớp chính xác là 78,50%, chứng minh hiệu quả và độ chính xác của nó trong các tác vụ tạo mã.

Tóm lại, nghiên cứu của chúng tôi đã tiết lộ một loạt các mô hình mã hóa-giải mã mạnh mẽ để tạo mã, mỗi mô hình đều thể hiện những điểm mạnh riêng. RoBERTaMarian và BERTMarian nổi bật với hiệu suất cạnh tranh của chúng trên tập dữ liệu CoNaLa, trong khi ELECTRAMarian và LUKEMarian cho thấy trình độ ấn tượng trên tập dữ liệu DJANGO. Ngoài ra, MarianCG đóng vai trò là cơ sở mạnh mẽ, mang lại kết quả đầy hứa hẹn trên cả hai tập dữ liệu. Hiểu được sự đánh đổi giữa các mô hình này cho phép chúng ta điều chỉnh ứng dụng của chúng cho nhiều tình huống tạo mã khác nhau, đáp ứng các yêu cầu cụ thể của từng tác vụ.

Những phát hiện này mở đường cho những tiến bộ hơn nữa trong công nghệ tạo mã và khả năng tích hợp của nó vào các ứng dụng lập trình và tự động hóa trong thế giới thực. Bảng 5.5 cho thấy kết quả và số liệu hiệu suất của các mô hình được đề xuất của chúng tôi trên tập dữ liệu CoNaLa. Ngoài ra, Bảng 5.6 cũng cho thấy kết quả của các mô hình được đề xuất đó trên tập dữ liệu DJANGO.

Table 5.5: Performance metrics of all models on the CoNaLa Dataset

No.	Model	Performance Metrics			
		Bleu Score	Exact Match Accuracy	Sacrebleu	Rouge Score
1	MarianCG Model	34.4291	10.2 %	30.6776	49.6272
2	RoBERTaMarian Model	35.7365	13.8 %	31.3025	44.2547
3	BERTMarian Model	32.4618	12.4 %	29.479	43.9467
4	ELECTRAMarian Model	30.1819	10.0 %	27.1495	42.4232
5	LUKEMarian Model	29.8281	7.6 %	25.3187	39.8431

Khi so sánh với các nhà nghiên cứu khác đã làm việc về vấn đề tạo mã trên CoNaLa, kết quả này có điểm BLEU cao nhất. Mục thứ hai cần đề cập là điểm ROUGE cao nhất hoặc kết quả ROUGE-L, cho biết số liệu thống kê dựa trên LCS. Vấn đề chuỗi con chung dài nhất có điểm tương đồng về cấu trúc ở cấp độ câu

Table 5.6: Performance metrics of all models on the DJANGO Dataset

No.	Model	Performance Metrics			
		Bleu Score	Exact Match Accuracy	Sacrebleu	Rouge Score
1	MarianCG Model	90.41	81.83 %	75.906	94.647
2	RoBERTaMarian Model	88.9123	77.95 %	74.083	92.7351
3	BERTMarian Model	56.55	76.676 %	64.884	88.692
4	ELECTRAMarian Model	53.02	65.319 %	58.155	83.905
5	LUKEMarian Model	89.3424	78.504 %	74.6583	93.1133

tính đến và tự động chọn n-gram đồng thời xuất hiện dài nhất theo trình tự. Mô hình MarianCG có điểm ROUGE cao nhất là 49,63.

Hơn nữa, Hình 5.6 chỉ ra kết quả của năm mô hình tạo mã được đề xuất. Các phát hiện cho thấy mô hình RoBERTaMarian tạo ra điểm BLEU cao nhất là 35,74. Ngoài ra, kết quả của các mô hình được đề xuất của chúng tôi được chỉ ra trong bảng 5.6 và được hiển thị trong Hình 5.7.

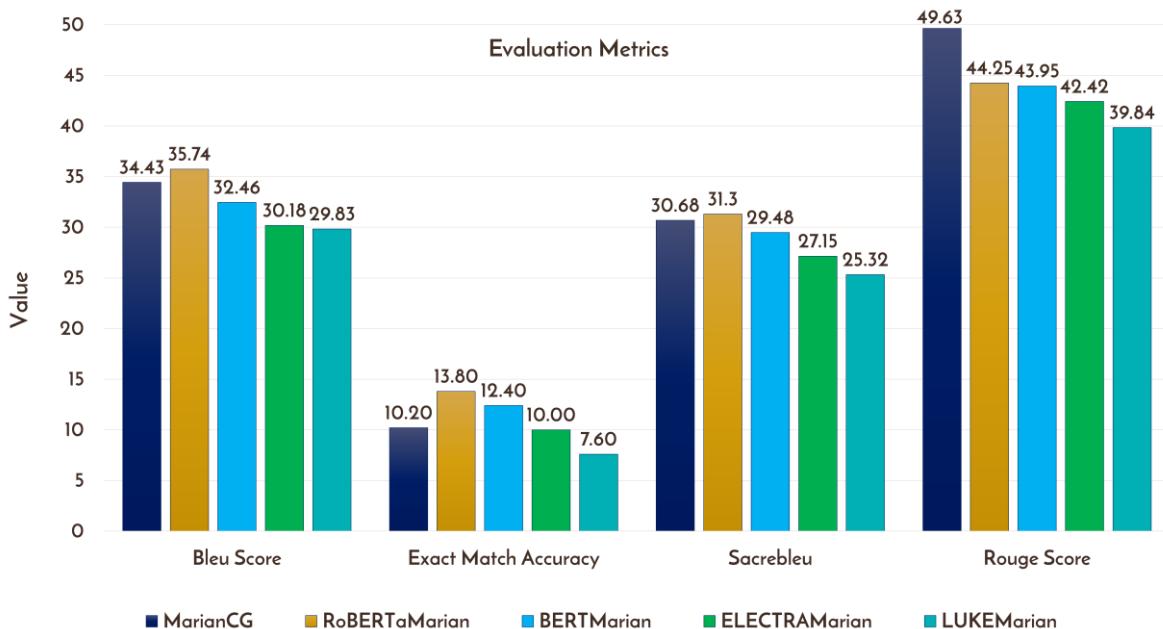


Figure 5.6: Results and Performance Metrics of our proposed models on the CoNaLa Dataset

Kết quả của các mô hình tạo mã hiện đại trên tập dữ liệu CoNaLa được thể hiện trong bảng 5.7 và được chỉ ra trong Hình 5.8 cho các mô hình CoNaLa.

RoBERTaMarian được coi là mô hình đầu tiên có kết quả chính xác về mã được tạo ra. Mô hình này ghi nhận điểm BLEU là 35,74 với ưu điểm là nhanh, nhỏ và chứa sự chú ý của nhiều đâu. Bên cạnh đó,

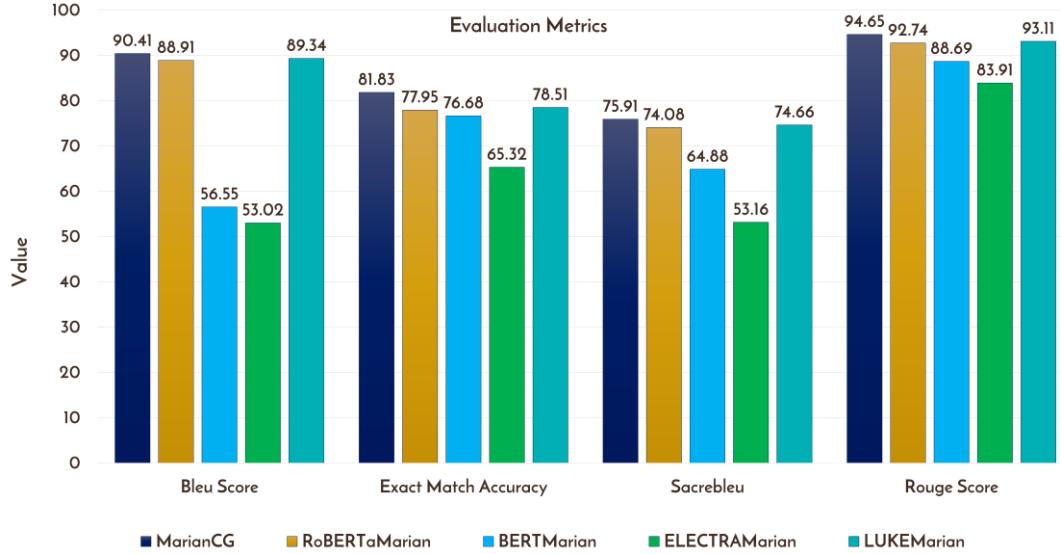


Figure 5.7: Results and performance metrics of our proposed models on the DJANGO dataset

Table 5.7: State of the art Code Generation models on CoNaLa

Rank	Model	BLEU	Exact Match Accuracy	Year
1	RoBERTaMarian (Ours)	35.7365	13.8 %	2023
2	MarianCG (Ours) [2]	34.4291	10.2 %	2022
3	TranX + BERT w/ mined [184]	34.2	5.8 %	2022
4	BERT + TAE [182]	33.41	-	2021
5	BERTMarian (Ours)	32.4618	12.4 %	2023
6	External Knowledge With API + Reranking [178]	32.26	-	2020
7	External Knowledge With API [178]	30.69	-	2020
8	BART W/ Mined [183]	30.55	-	2021
9	ELECTRAMarian (Ours)	30.1819	10.0 %	2023
10	Reranker [176]	30.11	-	2019
11	LUKE Marian (Ours)	29.8281	7.6 %	2023
12	BART Base [183]	26.24	-	2021
13	TranX [175]	24.3	-	2018

MarianCG được coi là mô hình thứ hai có điểm BLEU là 34,43. Đây là những mô

hình của chúng tôi ghi lại điểm đầu tiên và thứ hai trong kết quả mô hình tạo mã. Bên cạnh đó, các mô hình chúng tôi tạo ra có số điểm BLEU nằm trong các mô hình tạo mã xếp hạng cao nhất.

Ngoài ra, bảng 5.8 hiển thị kết quả của tất cả các mô hình tiên tiến trên DJANGO và các số liệu hiệu suất được chỉ ra trong Hình 5.9. So sánh tất cả các mô hình

hoạt động trên tập dữ liệu DJANGO đã chứng minh rằng các mô hình của chúng tôi có giá trị điểm BLEU cao như MarianCG, mô hình LUKEMarian và mô hình RoBERTaMarian. Các mô hình này có điểm BLEU lần lượt là 90,41, 89,34 và 88,91.

Table 5.8: State-of-the-art Code Generation models on DJANGO

Rank	Model	BLEU Score	Exact Match Accuracy	Year
1	MarianCG (Ours) [2]	90.41	81.83 %	2022
2	LUKEMarian (Ours)	89.3424	78.504 %	2023
3	RoBERTaMarian (Ours)	88.9123	77.95%	2023
4	TranX + BERT w/ mined [184]	79.86	81.03 %	2022
5	BERT + TAE [182]	-	81.77 %	2021
6	BERTMarian (Ours)	56.55	76.68 %	2023
7	ELECTRAMarian (Ours)	53.02	65.32 %	2022
8	Reranker [176]	-	80.2 %	2019
9	TranX [175]	-	73.7	2018

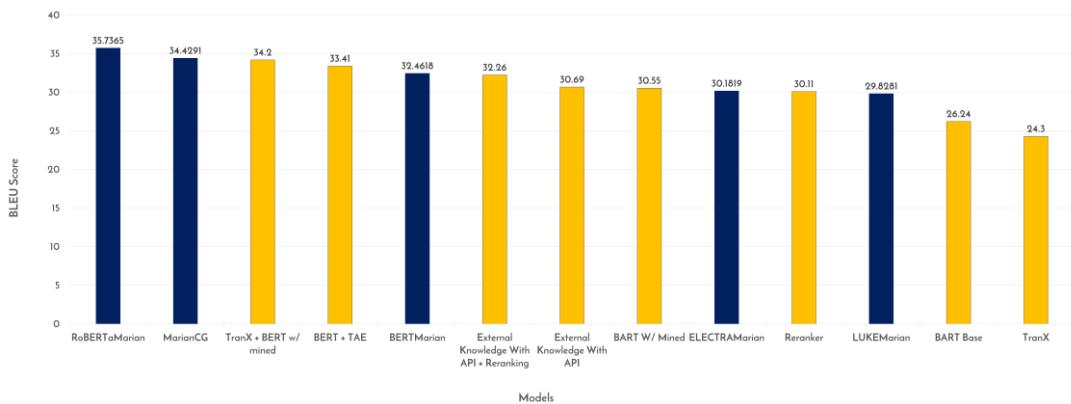


Figure 5.8: All Results of the state of the art models in the code generation problem with CoNaLa

Mô hình RoBERTaMarian là mô hình đầu tiên trong số các mô hình có kết quả chính xác về mã được tạo trên tập dữ liệu CoNaLa.

Ngoài ra, mô hình MarianCG được xếp hạng trong các mô hình hàng đầu về khả năng dự đoán chính xác về điểm BLEU và độ chính xác khớp chính xác. Mô hình này có kiến trúc kích thước nhỏ hơn. Nó có sáu lớp trong bộ mã hóa và sáu lớp trong bộ giải mã, trong khi các mô hình khác có kích thước mô hình lớn hơn.

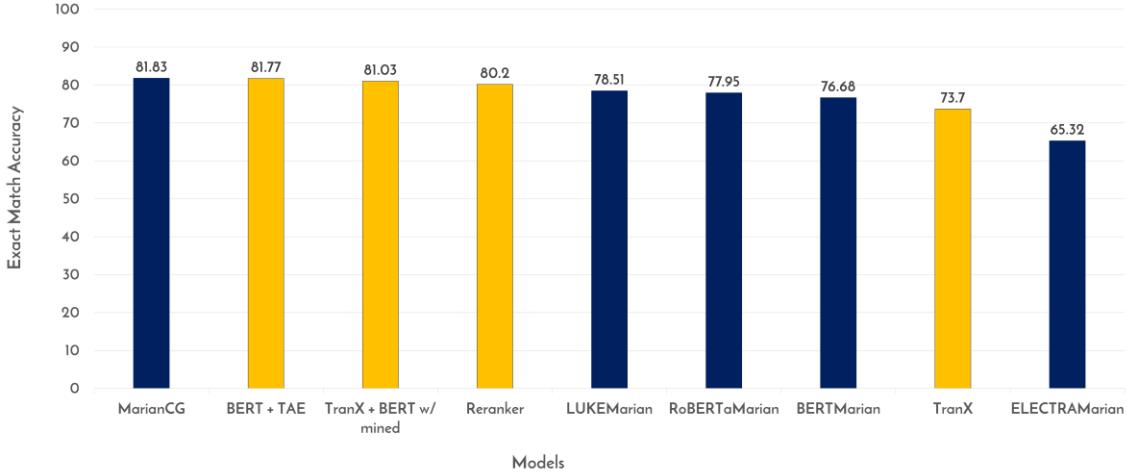


Figure 5.9: All results of the state-of-the-art models in the code generation on DJANGO with respect to Accuracy

5.5 Error/Warning Analysis and Refining

Trợ lý mã đại diện cho sự cộng sinh mạnh mẽ giữa các lập trình viên con người và công nghệ AI, cung cấp các công cụ thông minh hỗ trợ viết mã, phát hiện lỗi và tối ưu hóa[205] [206]. Bằng cách tận dụng các thuật toán AI và máy học, các nhà phát triển có thể nâng cao năng suất, cải thiện chất lượng mã và tạo điều kiện chia sẻ kiến thức trong các nhóm phát triển. Khi khả năng của các công cụ mã hóa hỗ trợ AI phát triển, chúng có tiềm năng biến đổi bối cảnh phát triển phần mềm, giúp quy trình hiệu quả hơn, đáng tin cậy hơn và mang tính cộng tác hơn.

Trong chương kết quả này, chúng tôi đã tiến hành phân tích lỗi và cảnh báo toàn diện bằng Flake8, một công cụ mạnh mẽ để kiểm tra lỗi mã và phân tích tĩnh. Flake8 đã giúp xác định các vấn đề tiềm ẩn trong mã Python được tạo, chẳng hạn như lỗi cú pháp, biến chưa được xác định và vi phạm kiểu mã. Phân tích lỗi và phân phối của mã do mô hình MarianCG tạo ra được hiển thị trong Bảng 5.9. Ngoài ra, Hình 5.10 hiển thị phân phối lỗi và cảnh báo trong mã được tạo ra.

Phân tích lỗi và phân phối từ mô hình RoBERTaMarian được hiển thị trong Bảng 5.10. Ngoài ra, Hình 5.11 cho thấy sự phân bố lỗi và cảnh báo trong mã được tạo ra.

Table 5.9: Error And Warning Analysis on The MarianCG Model

Error code	Error/warning message	Count
F821	"undefined name"	295
W292	"no newline at end of file"	230
F523	"'...'.format(...) has unused arguments at position(s): 1"	1
E999	"SyntaxError: invalid character"	270
E741	"ambiguous variable name 'l'"	2
E231	"missing whitespace after ',' or ':'"	5
E201	"whitespace after '('"	1

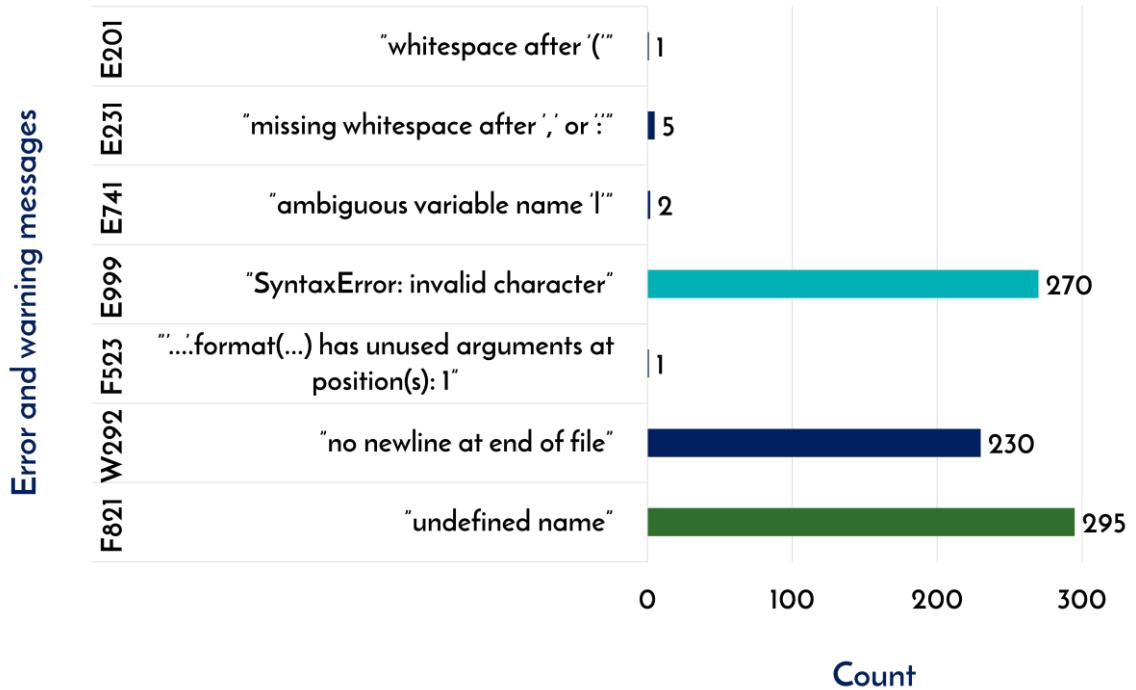


Figure 5.10: Error Analysis in the MarianCG Model Generated Code

Để giải quyết những lo ngại này và cải thiện chất lượng mã, chúng tôi đã sử dụng các công cụ định dạng mã tự động như Autopep8, tự động sửa các vấn đề liên quan đến thuật lê, độ dài dòng và khoảng trắng. Điều này dẫn đến mã sạch hơn và nhất quán hơn, phù hợp với các hướng dẫn về phong cách đã thiết lập.

Ngoài Autopep8, chúng tôi đã sử dụng thư viện Add-trailing-comma để tăng cường khả năng đọc mã và giảm lỗi liên quan đến danh sách và từ điển Python. Bằng cách thêm dấu phẩy sau vào các cấu trúc dữ liệu này, chúng tôi đảm bảo định dạng nhất quán, đặc biệt là khi sửa đổi hoặc mở rộng chúng

trong quá trình tạo mã.

Table 5.10: Error And Warning Analysis Through RoBERTaMarian Model

Error code	Error/warning message	Count
F821	"undefined name"	593
W292	"no newline at end of file"	438
E999	"SyntaxError: invalid character"	62
E741	"ambiguous variable name 'l'"	3
E225	"missing whitespace around operator"	4
F706	"'return' outside function"	4
E231	"missing whitespace after ','"	3
E713	"test for membership should be 'not in'"	1

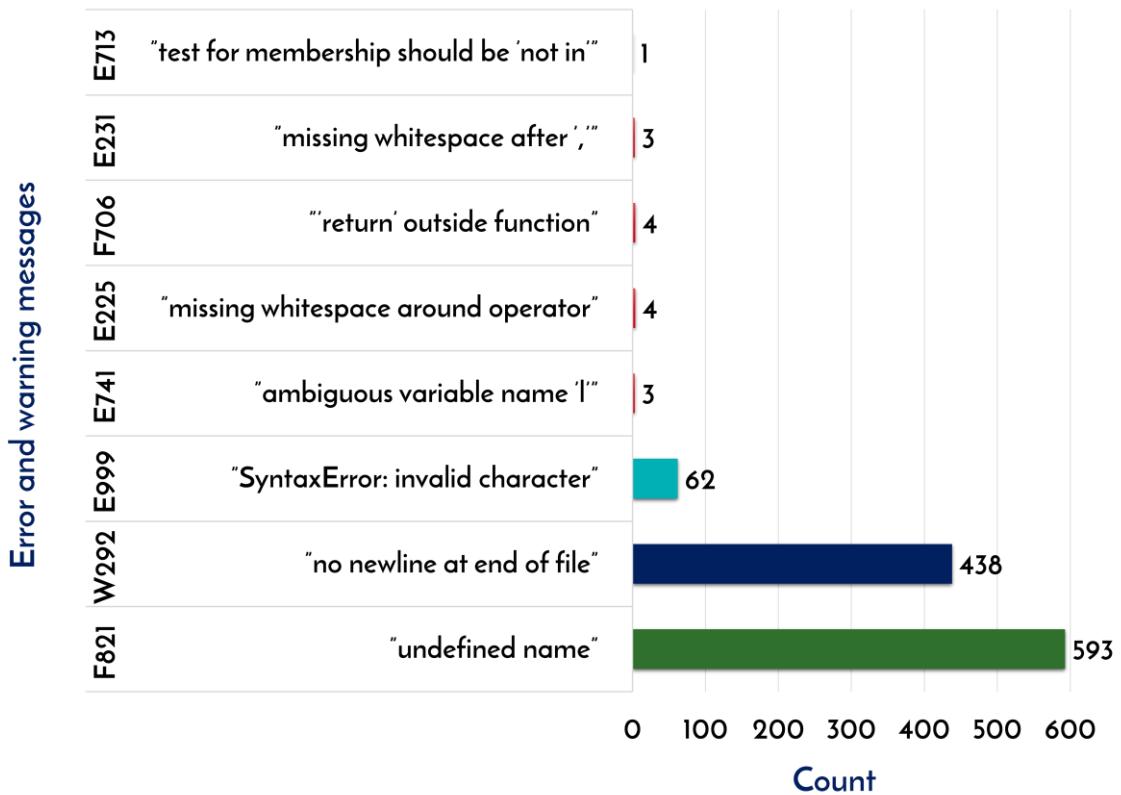


Figure 5.11: Error Analysis in the Generated Code using RoBERTaMarian Model

To maintain uniform code formatting across the generated solutions, we integrated two popular Python code formatters, Yapf and Black. Both tools automatically

Table 5.11: Error And Warning Analysis After Refining and Correction on MarianCG Model

Error code	Error/warning message	Count
F821	"undefined name"	295
F523	"'...'.format(...) has unused arguments at position(s): 1"	1
E999	"SyntaxError: invalid character"	64
E741	"ambiguous variable name 'l'"	2

formatted the code according to various style guides, such as PEP 8 and Google's style guide. Bằng cách sử dụng các trình định dạng này, khả năng đọc và bảo trì của mã được tạo ra được cải thiện đáng kể, giúp các nhà phát triển dễ hiểu và làm việc với mã hơn.

Yapf⁴, do Google phát triển, đã chứng minh khả năng cấu hình cao và giảm thiểu tác động của nó đối với mã được định dạng. Nó tuân thủ nghiêm ngặt hướng dẫn về phong cách PEP 8 trong khi nhấn mạnh vào khả năng đọc và định dạng nhất quán. Công cụ này cung cấp các tùy chọn linh hoạt để thiết lập độ dài dòng tối đa, phong cách đặt tên biến và thụt lề, cùng nhiều tùy chọn khác. Yapf tích hợp liền mạch với các công cụ phân tích mã Python khác, chẳng hạn như Flake8 và PyLint, để đảm bảo chất lượng mã toàn diện và nhất quán.

Để tổ chức các lần nhập hiệu quả, chúng tôi đã triển khai Isort, một thư viện Python có chức năng sắp xếp và định dạng hiệu quả các câu lệnh nhập trong mã Python. Bằng cách sử dụng Isort, chúng tôi duy trì thứ tự hợp lý và nhất quán cho các lần nhập, do đó nâng cao khả năng tổ chức và khả năng đọc của mã.

Cuối cùng, chúng tôi đã kết hợp Ruff, một thư viện Python mạnh mẽ để tái cấu trúc mã, nhằm nâng cao hơn nữa khả năng đọc và bảo trì của mã được tạo ra. Ruff tự động thực hiện các chuyển đổi mã, chẳng hạn như đơn giản hóa các biểu thức phức tạp, trích xuất các hàm và áp dụng nhiều kỹ thuật tái cấu trúc khác nhau, giúp cải thiện cấu trúc và độ rõ ràng của mã. Sau khi sử dụng các công cụ này, phân tích lỗi và cảnh báo trên mô hình MarianCG trở nên ít hơn trước như thể hiện trong Bảng 5.11 và phân phối lỗi được thể hiện trong Hình 5.12.

Sau khi sử dụng các công cụ tái cấu trúc này, phân tích lỗi và cảnh báo trên mô hình RoBER-TaMarian trở nên ít hơn trước như thể hiện trong Bảng 5.12 và phân phối lỗi được thể hiện trong Hình 5.13. Bằng cách tận dụng các công cụ và thư viện này trong quy trình tạo mã, chúng tôi đã đạt được các giải pháp mã không chỉ đáp ứng các tiêu chuẩn chất lượng cao mà còn được định dạng nhất quán và dễ bảo trì. Sự kết hợp của các công cụ phân tích tinh, kiểm tra lỗi, định dạng và tái cấu trúc đã góp phần đáng kể vào thành công chung của các mô hình tạo mã của chúng tôi và củng cố tầm quan trọng của chất lượng mã trong phát triển phần mềm hiện đại.

⁴<https://github.com/google/yapf>

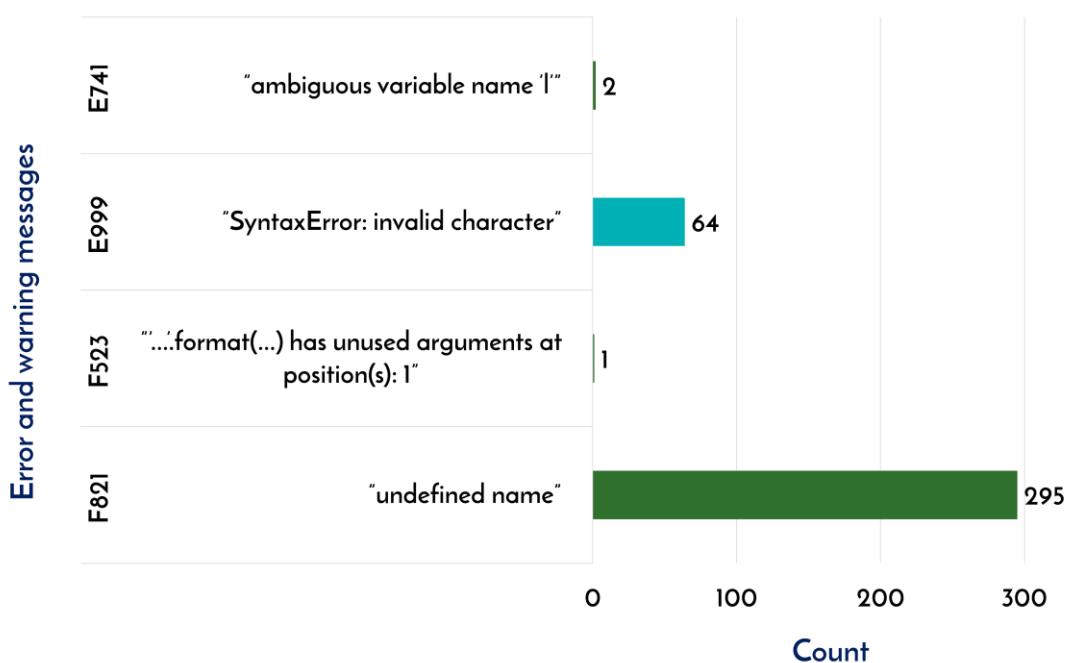


Figure 5.12: Refining and Correction of the Errors in the MarianCG Generated Code

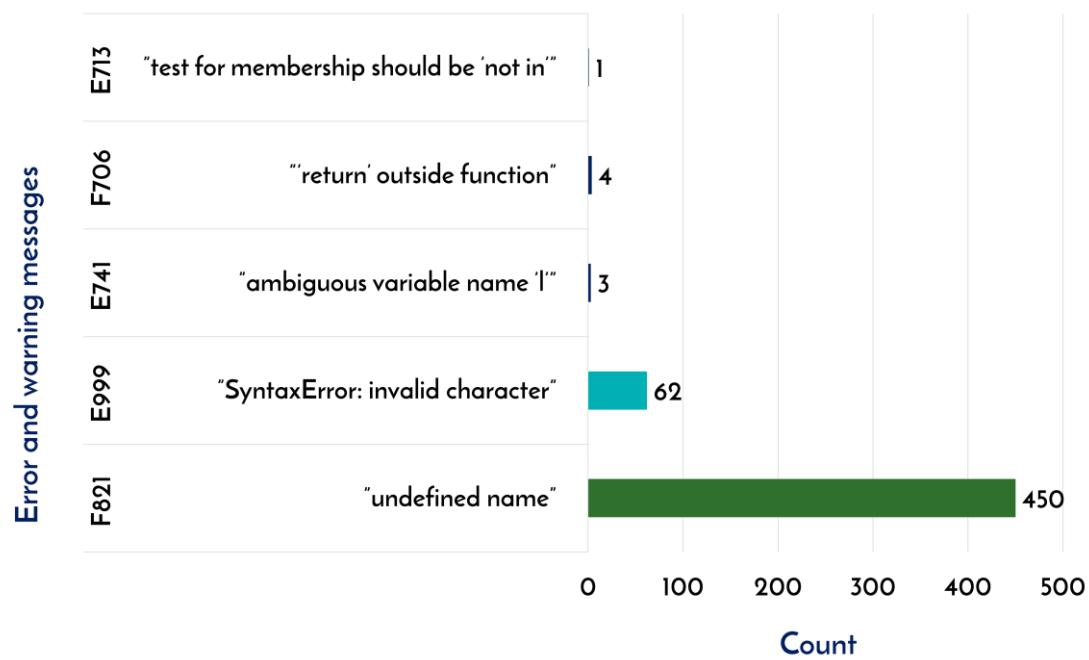


Figure 5.13: Refining and Correction of the Errors in the RoBERTaMarian Generated Code

Table 5.12: Error And Warning Analysis After Refining and Correction on RoBERTaMarian Model

Error code	Error/warning message	Count
F821	"undefined name"	450
E999	"SyntaxError: invalid character"	62
E741	"ambiguous variable name 'l'"	3
F706	"'return' outside function"	4
E713	"test for membership should be 'not in'"	1

Chapter 6

Multiline Code Generation with LLM

6.1 Introduction

Dựa trên những hiểu biết rút ra từ các mô hình trước đây của chúng tôi trong việc tạo mã, rõ ràng là trong khi các mô hình tạo mã một dòng cho thấy kết quả khá quan, chúng không miễn nhiễm với lỗi. Theo quan điểm này, chúng tôi chuyển sự chú ý của mình sang việc khám phá tiềm năng của các Mô hình ngôn ngữ lớn (LLM) và khả năng thích ứng của chúng thông qua việc tích hợp Thích ứng bậc thấp của các mô hình ngôn ngữ lớn (LoRA) [207] với các tham số lượng tử hóa, một kỹ thuật mới được gọi là QLoRA [208].

Ngoài ra, chúng tôi sử dụng phương pháp Điều chỉnh hiệu quả tham số (PEFT) [209] để nâng cao hơn nữa khả năng thích ứng và hiệu suất của các mô hình này. Sự kết hợp của LoRA [207] và các tham số lượng tử hóa cho phép điều chỉnh hiệu quả các LLM cho các tác vụ tạo mã cụ thể, giảm yêu cầu về tài nguyên và giúp chúng dễ tiếp cận hơn để triển khai thực tế.

Hơn nữa, chúng tôi nhận ra tầm quan trọng của việc đào tạo các mô hình này trên các tập dữ liệu tạo mã đa dòng, chẳng hạn như CodeSearchNet [193]. Bằng cách tận dụng dữ liệu tạo mã nhiều lớp, các mô hình sẽ tiếp xúc với các cấu trúc mã phức tạp, tăng cường khả năng tạo ra các giải pháp mã mạnh mẽ hơn và phù hợp với ngữ cảnh hơn. Việc tích hợp CodeSearchNet và các phương pháp QLoRA và PEFT tiên tiến mở ra những hướng đi mới để tạo ra các mô hình tạo mã có độ chính xác cao và hiệu quả, có thể đáp ứng nhiều tình huống mã hóa khác nhau.

Bằng cách giới thiệu LoRA, các tham số lượng tử hóa và PEFT vào lĩnh vực tạo mã, chúng tôi muốn giải quyết những hạn chế được quan sát thấy trong các mô hình tạo mã một dòng và mở đường cho các giải pháp hiệu quả và hiệu suất hơn. Sự kết hợp của LLM với các kỹ thuật mới này có tiềm năng cách mạng hóa các hoạt động tạo mã, giúp chúng thích ứng hơn, tiết kiệm tài nguyên hơn và dễ tiếp cận hơn với nhiều nhà phát triển và người dùng hơn.

Chương này đóng vai trò là một cuộc khám phá quan trọng về những tiến bộ đã đạt được trong công nghệ tạo mã, mở ra cánh cửa cho những khả năng thú vị cho nghiên cứu trong tương lai và triển khai thực tế trong lĩnh vực lập trình.

Sử dụng Mô hình ngôn ngữ lớn (LLM) tự hồi quy thay vì mô hình mã hóa-giải mã nhỏ trong các tác vụ tạo mã mang lại một số lợi thế có thể tác động đáng kể đến hiệu suất và chất lượng mã.

Đầu tiên, LLM, chẳng hạn như GPT-3 và BERT, được đào tạo trước trên lượng lớn dữ liệu văn bản đa dạng, giúp chúng có khả năng hiểu ngôn ngữ tự nhiên rất thành thạo. Quá trình đào tạo trước này cho phép LLM nắm bắt các mẫu ngôn ngữ phức tạp, cấu trúc cú pháp và thông tin theo ngữ cảnh, cho phép chúng tạo ra các đầu ra mã phù hợp hơn về mặt ngữ cảnh và có ý nghĩa ngữ nghĩa hơn.

Thứ hai, sử dụng LLM làm bộ giải mã giúp tiết kiệm tài nguyên tính toán và bộ nhớ so với mô hình mã hóa-giải mã đầy đủ. LLM chỉ tập trung vào giải mã văn bản đầu vào và tạo mã, giúp giảm gánh nặng tính toán so với xử lý song hướng cần thiết trong mô hình mã hóa-giải mã đầy đủ. Điều này giúp giải mã dựa trên LLM tiết kiệm thời gian và bộ nhớ hơn, giúp tạo mã khả thi ngay cả trên các thiết bị có tài nguyên hạn chế.

Ngoài ra, LLM có khả năng tạo mã từ các mô tả ngôn ngữ tự nhiên một phần hoặc không đầy đủ. Vì chúng được đào tạo trước trên nhiều tác vụ ngôn ngữ khác nhau, nên chúng có thể lắp đầy khoảng trống và tạo mã có liên quan ngay cả khi mô tả không đầy đủ hoặc mơ hồ. Khả năng thích ứng này có lợi cho các tác vụ tạo mã, trong đó các truy vấn ngôn ngữ tự nhiên có thể mơ hồ hoặc không chính xác.

Hơn nữa, việc sử dụng LLM hồi quy tự động cho phép có kiến trúc đơn giản và trực tiếp hơn. Bộ mã hóa chịu trách nhiệm xử lý đầu vào, trong khi bộ giải mã LLM tập trung vào việc tạo mã, tạo ra thiết kế mô hình hợp lý. Sự đơn giản này tạo điều kiện cho việc đào tạo mô hình dễ dàng hơn, suy luận nhanh hơn và khả năng diễn giải tốt hơn của mã được tạo.

Tuy nhiên, điều quan trọng cần lưu ý là việc chỉ sử dụng LLM làm bộ giải mã có thể không phù hợp với mọi tình huống tạo mã. Một số tác vụ có thể yêu cầu ngữ cảnh hai chiều và tương tác phức tạp hơn giữa bộ mã hóa và bộ giải mã. Trong những trường hợp như vậy, các mô hình bộ mã hóa-giải mã nhỏ có thể phù hợp hơn.

Vì vậy, việc sử dụng Mô hình ngôn ngữ lớn làm mô hình hồi quy tự động trong các tác vụ tạo mã mang lại một số lợi ích, bao gồm khả năng hiểu ngôn ngữ mạnh mẽ, hiệu quả tính toán, khả năng thích ứng với các truy vấn không đầy đủ và kiến trúc mô hình đơn giản hóa. Những lợi thế này khiến giải mã dựa trên LLM trở thành một phương pháp có giá trị để tạo ra mã được tạo ra có chất lượng cao và phù hợp với ngữ cảnh[210].

Trong chương này, chúng tôi đi sâu vào sự phức tạp của việc triển khai QLoRA và PEFT kết hợp với các kỹ thuật AI tạo ra trong LLM, thảo luận về lý thuyết của chúng về nền tảng LLM và những hàm ý thực tế mà chúng mang lại cho lĩnh vực tạo mã. Ngoài ra, thu thập kỹ thuật và điều chỉnh nhanh chóng với Llama Adapter để đào tạo hiệu quả hơn và lượng tử hóa. Chúng tôi khám phá tác động của chúng đối với khả năng thích ứng của mô hình, hiệu quả tài nguyên và hiệu suất khi so sánh với các mô hình tạo mã một dòng truyền thống.

Hình 6.1 cho thấy quy trình từ tinh chỉnh bộ điều hợp đến triển khai mô hình. Ngoài ra, chúng tôi trình bày kết quả đào tạo các mô hình này trên các tập dữ liệu tạo mã đa dòng, thể hiện sức mạnh của chúng trong việc tạo ra đầu ra mã đa dòng

và giải quyết các thách thức mã hóa trong thế giới thực với độ chính xác và độ chính xác cao hơn.



Figure 6.1: Process from fine-tuning an adapter to model deployment

6.2 Fine-tuning Large Language Models(LLMs)

Các Mô hình Ngôn ngữ Lớn (LLM) trải qua quá trình đào tạo trước trên một tập hợp văn bản mở rộng và trong trường hợp cụ thể của Llama 2[211], thông tin về thành phần chính xác của tập đào tạo bị hạn chế, với độ dài chỉ là 2 nghìn tỷ mã thông báo được biết đến. Để so sánh, mô hình BERT trước đó (2018) đã được đào tạo trên BookCorpus (800 triệu từ) và Wikipedia tiếng Anh (2.500 triệu từ). Đào tạo trước LLM liên quan đến việc đầu tư đáng kể về cả thời gian và nguồn lực, thường gặp phải những thách thức về phần cứng. Đối với những ai quan tâm đến việc tìm hiểu sâu hơn về quá trình đào tạo trước, tôi khuyên bạn nên tham khảo sổ ghi chép của Meta nêu chi tiết về quá trình đào tạo trước của mô hình OPT-175B.

Sau khi giai đoạn đào tạo trước của LLM như Llama 2 hoàn tất, các mô hình hồi quy tự động này có khả năng dự đoán các mã thông báo tiếp theo theo trình tự. Tuy nhiên, khả năng dự đoán này không biến chúng thành trợ lý hiệu quả, vì chúng không có khả năng phản hồi trực tiếp với các hướng dẫn cụ thể. Để thu hẹp khoảng cách này, điều chỉnh hướng dẫn được sử dụng để cẩn chỉnh các câu trả lời của mô hình với kỳ vọng của con người. Có hai kỹ thuật tinh chỉnh chính được sử dụng cho mục đích này:

1. Tinh chỉnh có giám sát (SFT): Trong SFT, các mô hình trải qua quá trình đào tạo bằng cách sử dụng một tập dữ liệu chứa các cặp hướng dẫn và phản hồi tương ứng. Trong quá trình này, trọng số của LLM được điều chỉnh để giảm thiểu sự chênh lệch giữa các câu trả lời được tạo ra và các phản hồi thực tế, sử dụng hiệu quả các phản hồi thực tế làm nhãn để hướng dẫn quá trình học của mô hình.

2. Học tăng cường từ phản hồi của con người (RLHF): Trong RLHF, các mô hình được đào tạo thông qua tương tác với môi trường của chúng và nhận phản hồi. Chúng được khuyến khích tối đa hóa tín hiệu phản thưởng, thường bắt nguồn từ các đánh giá của con người về đâu ra của mô hình. Tối ưu hóa chính sách gần (PPO) thường được sử dụng làm thuật toán tối ưu hóa cho phương pháp học tăng cường này.

Sự kết hợp của các kỹ thuật tinh chỉnh này cho phép các LLM như Llama 2 được chuyển đổi thành trợ lý hiệu quả hơn và có nhận thức theo ngữ cảnh, căn chỉnh các phản hồi của chúng với kỳ vọng của con người và các hướng dẫn cụ thể. Quá trình tinh chỉnh đóng vai trò quan trọng trong việc điều chỉnh các mô hình ngôn ngữ này cho các ứng dụng thực tế, nâng cao tiện ích của chúng như các công cụ có giá trị trong nhiều tác vụ Xử lý dựa trên Biến áp.

Trong chương này, chúng tôi đã tiến hành tinh chỉnh trên mô hình Llama 2 bao gồm 7 tỷ tham số bằng cách sử dụng tập dữ liệu CodeSearchNet. Quá trình tinh chỉnh được thực hiện trên GPU T4 có dung lượng RAM lớn, sử dụng Google Colab Pro để tăng cường sức mạnh tính toán. Tuy nhiên, điều cần lưu ý là GPU T4 cung cấp VRAM hạn chế, chỉ có 16 GB khả dụng, hầu như không đủ để lưu trữ trọng số lớn của Llama 2-7b (7 tỷ tham số, mỗi tham số được biểu thị bằng 2 byte trong FP16, tổng cộng là 14 GB). Ngoài ra, chúng tôi phải tính đến chi phí phát sinh từ các trạng thái tối ưu hóa, độ dốc và kích hoạt chuyển tiếp, càng làm cho quá trình tinh chỉnh trở nên khó khăn hơn do hạn chế về tài nguyên.

6.3 PEFT, LoRA and QLoRa

PEFT, LoRa và QLoRa là ba kỹ thuật cải tiến đã cách mạng hóa nhiều lĩnh vực nghiên cứu khác nhau. Chúng ta hãy cùng tìm hiểu sâu hơn về từng kỹ thuật và khám phá ý nghĩa của chúng. PEFT, viết tắt của Parameter Efficient Fine Tuning, là một phương pháp được sử dụng trong các mô hình học máy và học sâu [212]. PEFT tập trung vào việc tối ưu hóa các quy trình tinh chỉnh trong khi giảm thiểu số lượng tham số liên quan. Bằng cách tinh chỉnh và điều chỉnh hiệu quả các tham số, PEFT cho phép các mô hình đạt được hiệu suất được cải thiện mà không làm tăng đáng kể độ phức tạp của chúng. Kỹ thuật này có khả năng giảm yêu cầu tính toán và nâng cao hiệu quả tổng thể của các mô hình học máy.

Chuyển sang LoRa, còn được gọi là Low Rank Adaptation of Large Language Models, kỹ thuật này được ứng dụng trong nhiều tác vụ khác nhau. LoRa giải quyết thách thức trong việc điều chỉnh và tinh chỉnh các mô hình ngôn ngữ lớn cho các tác vụ cụ thể trong khi tránh tình trạng quá khớp và duy trì hiệu quả tính toán. Bằng cách tận dụng các kỹ thuật phân tích ma trận bậc thấp, LoRa tối ưu hóa quy trình điều chỉnh các mô hình ngôn ngữ này, mang lại hiệu suất và khả năng khai quát hóa tốt hơn trên nhiều tác vụ NLP khác nhau.

LoRa thêm một lượng nhỏ các tham số có thể đào tạo, tức là bộ điều hợp, cho mỗi lớp của LLM và đóng băng tất cả các tham số gốc. Để tinh chỉnh, chúng ta chỉ cần cập nhật trọng số bộ điều hợp, giúp giảm đáng kể dấu chân bộ nhớ.

Cuối cùng, chúng ta có QLoRa, viết tắt của Quantized LoRA. QLoRa là một tiến bộ trong LoRa tập trung vào việc tối ưu hóa hơn nữa quá trình tinh chỉnh bằng cách kết hợp các kỹ thuật lượng tử hóa như thể hiện trong Hình 6.2. Lượng tử hóa liên quan đến việc giảm độ chính xác hoặc độ rộng bit của các biểu diễn số, tạo ra các mô hình

nhỏ gọn hơn và hiệu quả hơn về mặt tính toán[213].

Bằng cách kết hợp các lợi ích của khả năng thích ứng cấp thấp của LoRa với các kỹ thuật lượng tử hóa, QLoRa cung cấp một phương pháp mới để tinh chỉnh các mô hình ngôn ngữ lớn với bộ nhớ và yêu cầu tính toán giảm. Hình 6.3 và Hình 6.4 cho thấy cách tinh chỉnh mô hình bằng PEFT và LoRA và không sử dụng tất cả các trọng số và tinh chỉnh thường xuyên.

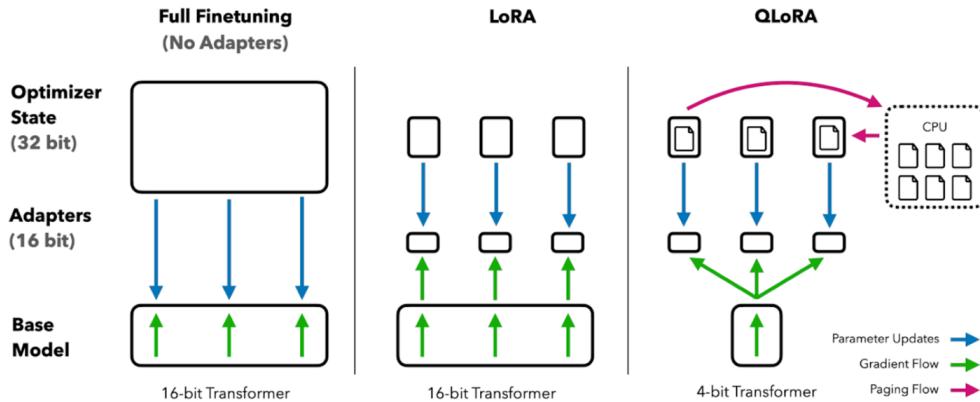


Figure 6.2: Different finetuning methods and their memory requirements. QLORA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.[208]

Tóm lại, PEFT, LoRa và QLoRa đại diện cho những tiến bộ đáng kể trong các lĩnh vực tương ứng của chúng. Trong khi PEFT tập trung vào hiệu quả tham số và tinh chỉnh trong học máy, LoRa và QLoRa giải quyết các thách thức liên quan đến việc điều chỉnh và tối ưu hóa các mô hình ngôn ngữ lớn. Các kỹ thuật này mở ra những khả năng mới để cải thiện hiệu suất, hiệu quả và khả năng thích ứng trên nhiều miền khác nhau.

6.4 Llama2

Meta đã phát triển và phát hành họ Llama 2 gồm các mô hình ngôn ngữ lớn (LLM)[211], bao gồm các mô hình văn bản tạo ra được đào tạo trước và tinh chỉnh có quy mô từ 7 tỷ đến 70 tỷ tham số. Trong số các mô hình này, Llama-2-Chat được tối ưu hóa cụ thể cho các trường hợp sử dụng đối thoại, vượt trội hơn các mô hình trò chuyện nguồn mở trong nhiều điểm chuẩn khác nhau và đạt được kết quả tương đương với các mô hình nguồn đóng phổ biến như ChatGPT và PaLM về tính hữu ích và an toàn dựa trên đánh giá của con người.

Họ Llama 2 cung cấp các biến thể về kích thước tham số, bao gồm 7B, 13B và 70B, mỗi biến thể đều trình bày các phương án được đào tạo trước và tinh chỉnh. Các mô hình này chỉ chấp nhận văn bản đầu vào và tạo văn bản làm đầu ra của chúng, sử dụng mô hình ngôn ngữ tự hồi quy dựa trên kiến trúc biến đổi được tối ưu hóa.

LoRA weights, W_A and W_B , represent ΔW

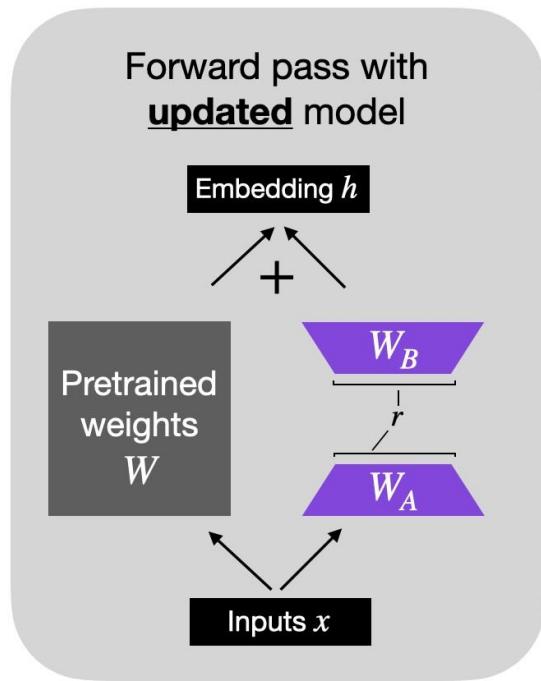


Figure 6.3: LoRA Weights in the fine-tuning using LoRA and PEFT[214]

Các phiên bản tinh chỉnh trải qua quá trình tinh chỉnh có giám sát (SFT) và học tăng cường với phản hồi của con người (RLHF) để phù hợp với sở thích của con người về tính hữu ích và an toàn.

Để đảm bảo hiệu suất tối ưu của các phiên bản trò chuyện, cần tuân theo các hướng dẫn định dạng cụ thể, bao gồm việc sử dụng thẻ INST và «SYS», mã thông báo BOS và EOS, cũng như khoảng trắng và đường ngắt thích hợp. Những cân nhắc về định dạng này giúp đạt được các tính năng và hiệu suất mong đợi trong các tình huống hướng đến trò chuyện.

Llama 2 đã được đào tạo trên một tập dữ liệu mở rộng gồm 2 nghìn tỷ mã thông báo từ các nguồn có sẵn công khai và dữ liệu tinh chỉnh bao gồm các tập dữ liệu hướng dẫn có sẵn công khai và hơn một triệu ví dụ mới do con người chú thích. Điều cần lưu ý là cả tập dữ liệu tiền đào tạo và tập dữ liệu tinh chỉnh đều không chứa dữ liệu người dùng Meta. Mô hình hiện đang ở trạng thái tĩnh và được đào tạo trên một tập dữ liệu ngoại tuyến, với các phiên bản trong tương lai của các mô hình được tinh chỉnh sẽ được phát hành.

Alternative formulation (regular finetuning)

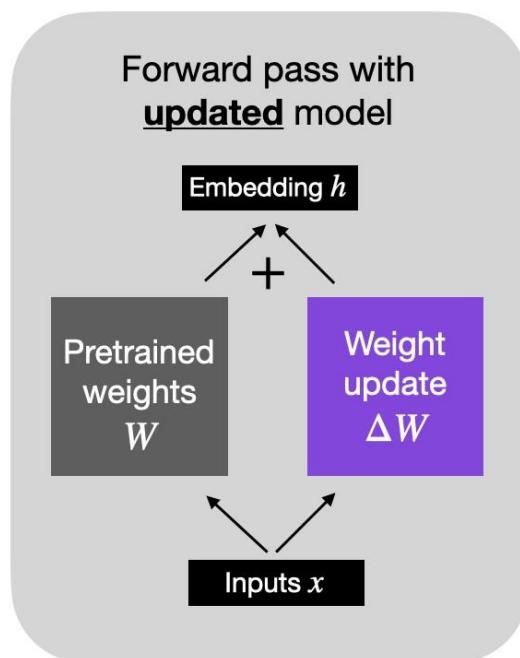


Figure 6.4: Regular Weights in the fine-tuning process[214]

6.5 Our Proposed Approach: Llama2-CodeGen

Trong công việc của mình, chúng tôi cần các thư viện tăng tốc, peft, transformers, datasets và TRL để tận dụng SFTTrainer gần đây. Chúng tôi sử dụng bitsandbytes để lượng tử hóa mô hình cơ sở thành 4bit. Chúng tôi cũng cài đặt einops. Mô hình Llama2-CodeGen1 là LlaMa2-7b được tinh chỉnh trên tập dữ liệu CodeSearchNet bằng phương pháp QLoRA với thư viện PEFT. Mô hình này được đào tạo trên Google Colab Pro bằng AutoTrain, PEFT và QLoRA. Bạn có thể tải mô hình LlaMa2-CodeGen trên Google Colab và triển khai trên GitHub².

Các kỹ thuật tinh chỉnh nâng cao LLM như Llama 2, biến chúng thành trợ lý hiệu quả hơn và có nhận thức theo ngữ cảnh, giúp phản hồi của chúng phù hợp với kỳ vọng của con người và hướng dẫn cụ thể. Quá trình tinh chỉnh này đóng vai trò quan trọng trong việc tùy chỉnh các mô hình ngôn ngữ để sử dụng thực tế, biến chúng thành công cụ có giá trị trong nhiều nhiệm vụ khác nhau.

Trong mô hình đề xuất của chúng tôi, chúng tôi đã tinh chỉnh mô hình Llama 2 với 7 tỷ tham số bằng cách sử dụng tập dữ liệu CodeSearchNet và quá trình tiền xử lý đối với tập dữ liệu này được thể hiện trong Hình 6.5. Quy trình tinh chỉnh diễn ra trên GPU T4 được trang bị dung lượng RAM lớn, tận dụng Google Colab Pro để tăng khả năng tính toán

¹<https://huggingface.co/AhmedSSoliman/Llama2-CodeGen-PEFT-QLoRA>

²<https://github.com/AhmedSSoliman/Llama2-CodeGen-Fine-Tuning-LLama-2>

power. Tuy nhiên, điều quan trọng là phải xem xét VRAM hạn chế của GPU T4, chỉ cung cấp 16 GB, chứng tỏ là không đủ để lưu trữ trọng số lớn của Llama 2-7b (7 tỷ tham số, mỗi tham số được biểu diễn bằng 2 byte trong FP16, tạo ra tổng cộng 14 GB). Ngoài ra, chúng tôi gặp phải những thách thức phát sinh từ chi phí liên quan đến trạng thái tối ưu hóa, độ dốc và kích hoạt chuyển tiếp, làm quá trình tinh chỉnh trở nên căng thẳng hơn nữa.

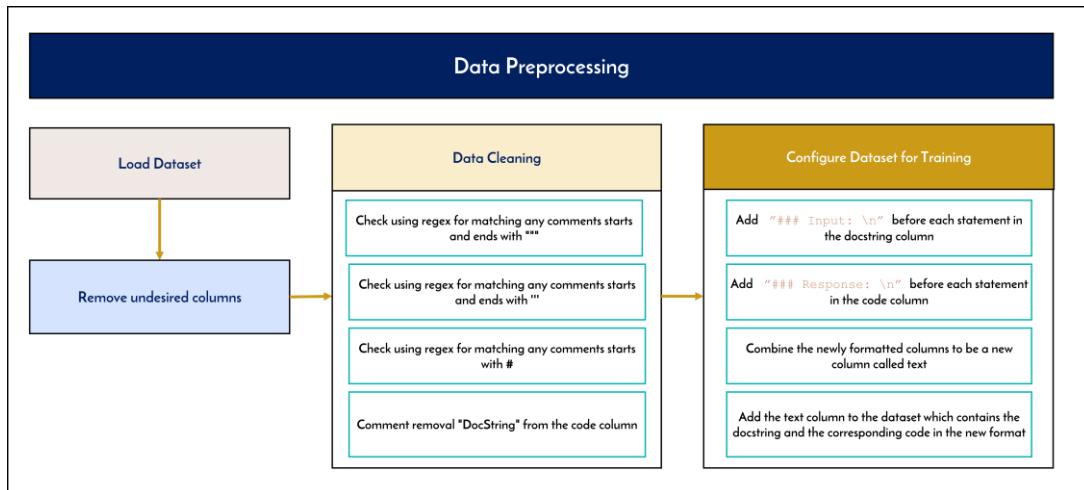


Figure 6.5: Data Preprocessing and Phases to Configure the Dataset

Chúng tôi đã làm việc bằng cách áp dụng các kỹ thuật tinh chỉnh hiệu quả tham số (PEFT) như LoRA hoặc QLoRA và Kỹ thuật nhắc nhở với Điều chỉnh bộ điều hợp như thể hiện trong Hình 6.6. Để giảm đáng kể mức sử dụng VRAM, chúng tôi đã chọn tinh chỉnh mô hình bằng độ chính xác 4 bit, khiến QLoRA trở thành phương pháp được ưu tiên cho mục đích này.

May mắn thay, chúng tôi đã khai thác các khả năng của hệ sinh thái Hugging Face, sử dụng nhiều thư viện khác nhau như transformers, accelerator, peft, trl và bitsandbytes để hợp lý hóa và đẩy nhanh quá trình tinh chỉnh. Các công cụ này đóng vai trò quan trọng trong việc nâng cao hiệu quả, cho phép chúng tôi thực hiện tinh chỉnh hiệu quả, bất chấp những thách thức do nguồn lực hạn chế đặt ra.

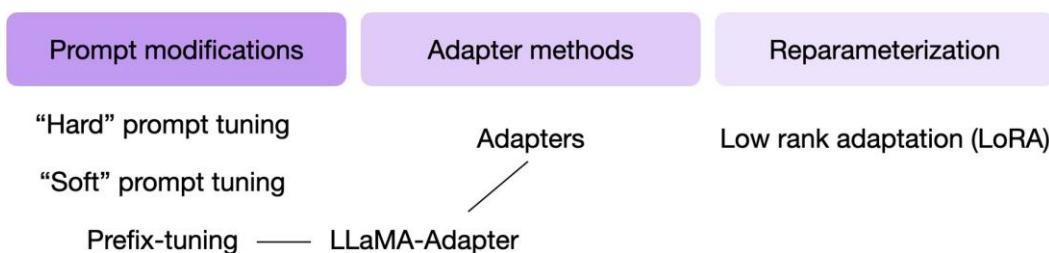


Figure 6.6: Fine-tuning Llama-2 with Prompt Tuning, Adapter Tuning and LoRA[215]

6.6 Llama2-CodeGen Example

Input: Write a program that adds five numbers.

Response:

```
def main():
    n1 = int(input("Enter the first number: "))
    n2 = int(input("Enter the second number: "))
    n3 = int(input("Enter the third number: "))
    n4 = int(input("Enter the fourth number: "))
    n5 = int(input("Enter the fifth number: "))

    sum = n1 + n2 + n3 + n4 + n5

    print("The sum of the five numbers is " + str(sum))

main()
```

Input: Write a Python code for reading multiple images from a path.

Response:

```
def read_multiple_images(path):
    images = []
    for image_name in os.listdir(path):
        image_path = os.path.join(path, image_name)
        if os.path.isfile(image_path):
            images.append(cv2.imread(image_path))
    return np.array(images)
```

6.7 Prompt Engineering And Tuning, Prefix Tuning, And Adapter Tuning

6.7.1 Prompt Tuning And Prefix Tuning

Khái niệm ban đầu về điều chỉnh nhở để cập đến các kỹ thuật thay đổi nhở nhào đầu vào để đạt được kết quả mô hình hóa tốt hơn. Điều này được gọi là điều chỉnh nhở cứng. Điều chỉnh nhở cứng chỉ đơn giản là thay đổi hoàn toàn nhở nhào đầu vào thành LLM. Điều chỉnh nhở mềm giới thiệu

các tham số có thể đào tạo được thêm vào nhúng đầu vào của mô hình.

Điều chỉnh nhắm mèm nối các nhúng của mã thông báo đầu vào với một tenxơ có thể đào tạo được có thể được tối ưu hóa thông qua truyền ngược để cải thiện hiệu suất mô hình hóa trên một tác vụ mục tiêu. Một hương vị cụ thể của điều chỉnh nhắm là điều chỉnh tiền tố[216]. Ý tưởng trong điều chỉnh tiền tố là thêm một tenxơ có thể đào tạo được vào mỗi khối biến áp thay vì chỉ nhúng đầu vào, như trong điều chỉnh nhắm mềm. Hình 6.7 minh họa sự khác biệt giữa khối biến áp thông thường và khối biến áp được sửa đổi bằng điều chỉnh tiền tố. Trong quá trình tinh chỉnh, các tham số ban đầu của LLM được giữ nguyên trong khi các tham số tiền tố được cập nhật.

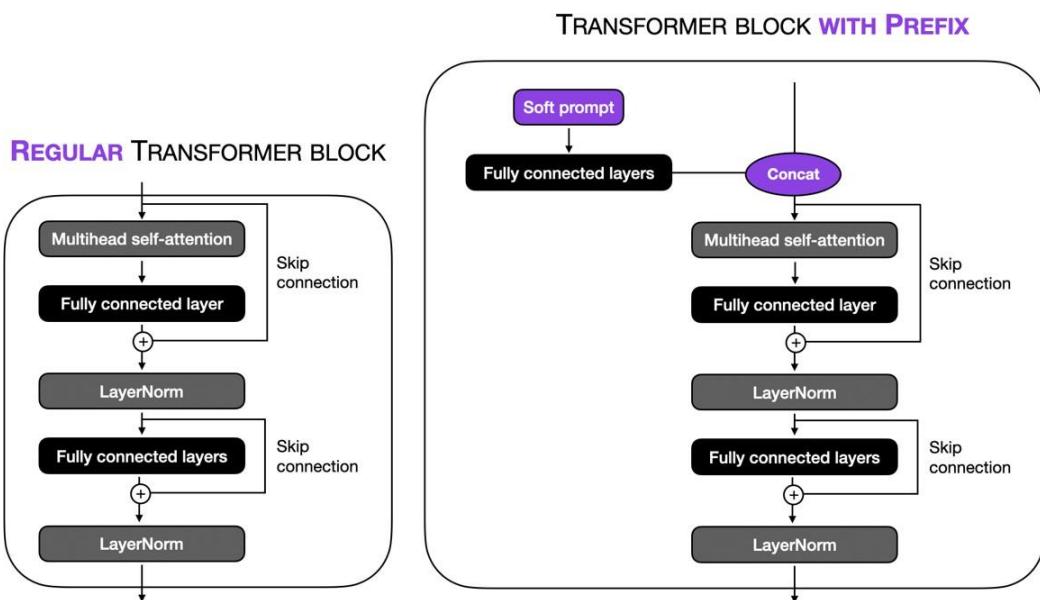


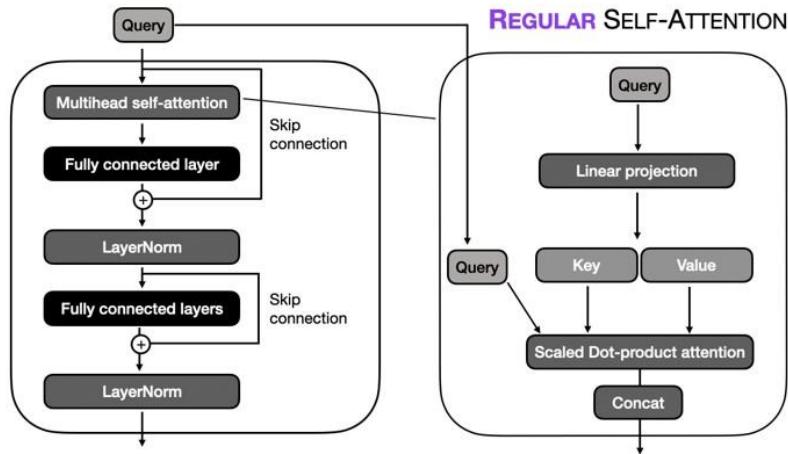
Figure 6.7: Prompt Tuning[215]

6.7.2 Adapter Tuning

Điều chỉnh dựa trên bộ điều hợp[217] chỉ cần chèn các mô-đun mới được gọi là "mô-đun bộ điều hợp" giữa các lớp của mạng được đào tạo trước. Trong Điều chỉnh bộ điều hợp, các tham số của mạng gốc được đóng băng và do đó có thể được chia sẻ bởi nhiều tác vụ. Mở rộng các ý tưởng về điều chỉnh tiền tố và phương pháp bộ điều hợp gốc, các nhà nghiên cứu gần đây đã đề xuất LLaMA-Adapter[218], một phương pháp điều chỉnh hiệu quả về tham số cho LLaMA[211] như thể hiện trong Hình.6.8.

Giống như điều chỉnh tiền tố, phương pháp Llama-Adapter thêm các tenxơ nhắm có thể điều chỉnh vào các đầu vào nhúng. Cần lưu ý rằng trong phương pháp LLaMA-Adapter, tiền tố được học và duy trì trong một bảng nhúng thay vì được cung cấp bên ngoài. Mỗi khối biến áp trong mô hình có tiền tố học riêng biệt, cho phép điều chỉnh phù hợp hơn trên các lớp mô hình khác nhau.

REGULAR TRANSFORMER BLOCK



TRANSFORMER BLOCK WITH LLAMA ADAPTER

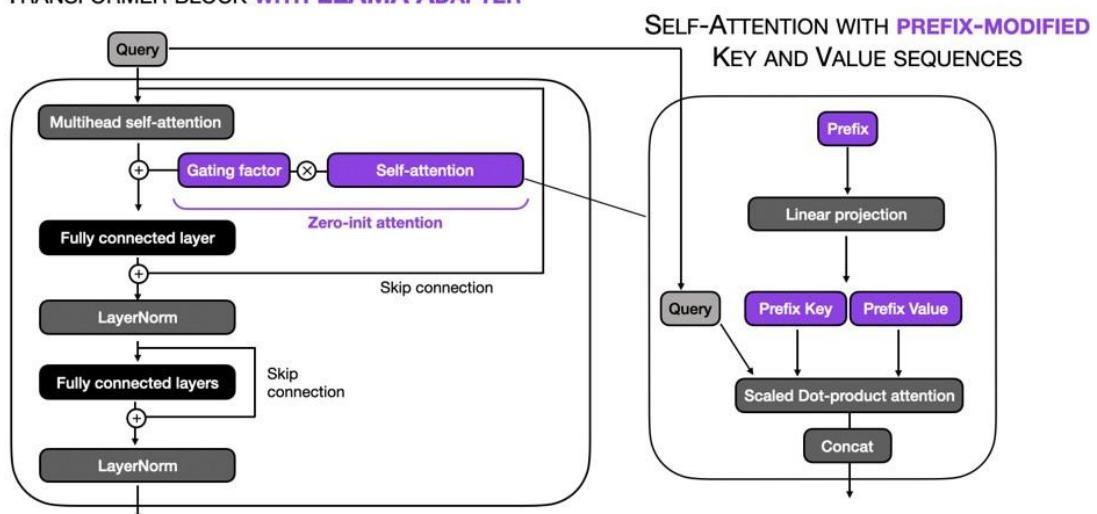


Figure 6.8: Llama-Adapter[215]

Chapter 7

Conclusion and Future Work

7.1 Conclusion

Tạo mã đã nổi lên như một khía cạnh quan trọng và hấp dẫn, giúp đơn giản hóa quá trình mã hóa cho cả lập trình viên có kinh nghiệm và người mới bắt đầu. Nghiên cứu này tập trung vào nhiệm vụ tạo mã để dịch các mô tả ngôn ngữ tự nhiên thành chuỗi mã. Tận dụng các kỹ thuật học sâu và đào tạo trước, chúng tôi đã phát triển thành công các kiến trúc nhỏ gọn với đầu ra chính xác, giám thiểu mức tiêu thụ bộ nhớ. Một cách tiếp cận hiệu quả để tạo mã liên quan đến việc sử dụng các mô hình ngôn ngữ được đào tạo trước, điều này đã được chứng minh là có lợi cho các mô hình của chúng tôi. Với các mô hình được đào tạo trước, các kiến trúc mã hóa-giải mã có thể được tạo và đào tạo hiệu quả với các trọng số được cập nhật, nâng cao hiệu suất và hiệu quả của chúng trong việc tạo mã từ các mô tả ngôn ngữ tự nhiên.

Một trong những cách tiếp cận hiệu quả nhất để tạo mã liên quan đến việc sử dụng các mô hình ngôn ngữ được đào tạo trước, đóng vai trò là nền tảng cho các đóng góp được đề xuất của chúng tôi cho các tập dữ liệu CoNaLa DJANGO và CodeSearchNet. Chúng tôi đã thiết kế và tinh chỉnh tỉ mỉ mô hình tạo mã của mình, có tên là MarianCG, dựa trên mô hình biến đổi MarianMT, dựa trên bộ công cụ Microsoft Marian. Mô hình MarianCG đã thể hiện những lợi thế đáng kể, bao gồm học không-shot, kiến trúc nhúng vị trí hình sin, cơ chế chú ý nhiều đâu và bộ phân tích Marian được xây dựng trên SentencePiece. Chúng tôi đã tiết lộ rằng các mô hình dịch máy không chỉ hoạt động như các mô hình tạo mã mà còn hoạt động tốt khi so sánh với các mô hình tạo mã khác.

Thông qua một loạt các thí nghiệm, chúng tôi đã đào tạo và thử nghiệm năm mô hình riêng biệt, mỗi mô hình kết hợp các mô hình ngôn ngữ được đào tạo trước với Marian Decoder và tùy thuộc vào Transformer-Based Processing. Trong thí nghiệm đầu tiên, mô hình MarianCG đạt điểm BLEU là 34,43 và độ chính xác khớp chính xác là 10,2% trên tập dữ liệu CoNaLa. Thí nghiệm thứ hai, được tiến hành trên tập dữ liệu DJANGO, đã mang lại kết quả đáng chú ý với điểm BLEU là 90,41 và độ chính xác khớp chính xác là 81,83%. Trong những nỗ lực tiếp theo, chúng tôi đã khám phá các mô hình ngôn ngữ được đào tạo trước khác nhau, bao gồm DistilRoBERTa, ELECTRA và LUKE, và tích hợp chúng với Marian Decoder để tạo ra các mô hình tạo mã. RoBERTaMarian đạt điểm BLEU cao nhất là 35,74 trên tập dữ liệu CoNaLa..

Phát hiện của chúng tôi chứng minh rằng các mô hình ngôn ngữ được đào tạo trước, khi kết hợp với Marian Decoder, đóng vai trò là giải pháp hiệu quả cho các tác vụ tạo mã. Hơn nữa, chúng tôi đã nêu bật sự tương đồng giữa các thách thức về tạo mã và dịch máy, trong đó MarianMT, một mô hình dịch máy mạnh mẽ, đã chứng tỏ là lựa chọn đặc biệt cho tất cả các mô hình tạo mã của chúng tôi.

Cuối cùng, chúng tôi đã làm việc để sử dụng Mô hình ngôn ngữ lớn Llama2 trong tác vụ tạo mã và tạo ra Llama2-CodeGen. Điều này được thực hiện bằng các phương pháp PEFT và QLoRA. Llama2-CodeGen mang lại một số lợi ích, bao gồm khả năng hiểu ngôn ngữ mạnh mẽ, hiệu quả tính toán, khả năng thích ứng với các truy vấn không đầy đủ và kiến trúc mô hình được đơn giản hóa. Những lợi thế này khiến giải mã dựa trên LLM trở thành một phương pháp có giá trị để tạo mã chất lượng cao và có liên quan theo ngữ cảnh từ các mô tả ngôn ngữ tự nhiên.

Tóm lại, nghiên cứu này đã chứng minh hiệu quả của các mô hình ngôn ngữ được đào tạo trước trong việc tạo mã và ứng dụng tiềm năng của các mô hình dịch máy để giải quyết tác vụ này. Mô hình MarianCG, cùng với mô hình RoBERTaMarian, nổi lên là những mô hình có hiệu suất cao nhất trong các thí nghiệm của chúng tôi. Sự thành công của các mô hình này mở đường cho những tiến bộ trong tương lai về công nghệ tạo mã, giúp việc lập trình dễ tiếp cận và hiệu quả hơn đối với cả lập trình viên và người mới bắt đầu.

7.2 Future Work

- Độ phức tạp của văn bản đầu vào là yếu tố quan trọng trong việc xác định độ phức tạp của tác vụ tạo mã. Khi văn bản đầu vào trở nên dài hơn, có cấu trúc hơn và chứa nhiều từ vựng chuyên ngành hơn, thì tác vụ trở nên ngày càng khó khăn hơn. Điều này là do mô hình cần nắm bắt các phụ thuộc tầm xa hơn, xử lý sự mơ hồ và tạo mã mạch lạc và chính xác đáp ứng các yêu cầu của tác vụ đã cho.
- Để định lượng độ phức tạp của công việc trong tương lai trong văn bản đầu vào, chúng ta có thể xem xét một số yếu tố như:
 - Độ dài câu: Các câu dài hơn với nhiều mệnh đề và cụm từ đòi hỏi cơ chế mã hóa và giải mã phức tạp hơn để nắm bắt mối quan hệ giữa các mã thông báo, thực thể và hành động.
 - Từ vựng chuyên ngành: Các lĩnh vực chuyên biệt như ngôn ngữ lập trình, tài liệu pháp lý hoặc báo cáo y tế thường chứa thuật ngữ chuyên ngành và thuật ngữ đòi hỏi phải hiểu sâu hơn về chủ đề.
 - Sự mơ hồ và không chắc chắn: Ngôn ngữ tự nhiên vốn có tính mơ hồ, với các từ và cụm từ có nhiều nghĩa và ngữ cảnh. Mô hình phải có khả năng giải

quyết những sự mơ hồ này và tạo mã phù hợp đáp ứng các ràng buộc của tác vụ.

- Độ phức tạp về mặt cấu trúc: Nhiệm vụ tạo mã có thể liên quan đến việc tạo mã bao gồm các cấu trúc lồng nhau, vòng lặp, điều kiện và hàm, đòi hỏi phải hiểu sâu về các khái niệm lập trình và cú pháp.

- Độ phức tạp của văn bản đầu vào ảnh hưởng trực tiếp đến độ phức tạp của mã kết quả. Ví dụ, một tập lệnh giao diện dòng lệnh (CLI) đơn giản có thể yêu cầu mã ít phức tạp hơn so với ứng dụng web có nhiều tuyến đường, xác thực người dùng và tương tác cơ sở dữ liệu. Tương tự như vậy, một đoạn mã thực hiện các phép toán số học cơ bản có thể đơn giản hơn đoạn mã triển khai các thuật toán học máy hoặc tính toán hiệu suất.

Nghiên cứu này chứng minh tiềm năng của các mô hình ngôn ngữ biến đổi được đào tạo trước trong quá trình tạo mã và nêu bật các cơ hội cho nghiên cứu trong tương lai về các chủ đề mới nổi như tạo mã đa phương thức, AI có thể giải thích được, các cuộc tấn công đối nghịch và sự hợp tác giữa con người và AI. Việc giải quyết các hạn chế của nghiên cứu và các tác động về mặt đạo đức sẽ giúp đảm bảo việc áp dụng có trách nhiệm các mô hình này trong quá trình phát triển phần mềm, cuối cùng là nâng cao năng suất và hiệu quả của các kỹ sư phần mềm.

Mục tiêu dài hạn của chúng tôi là phát triển các mô hình tạo mã có thể tạo mã nhiều dòng trong khi vẫn duy trì phương pháp tối ưu để phát triển chương trình. Tối ưu hóa mã hóa bao gồm việc giảm kích thước mã trong khi vẫn đảm bảo chức năng hiệu quả thực hiện cùng một nhiệm vụ. Tạo mã nhiều dòng sẽ là một cách tiếp cận tuyệt vời để tự động viết toàn bộ chương trình hoặc nhiều phần của mã và chứng minh cách tạo mã nguồn cho một ngôn ngữ lập trình nhất định từ một ngôn ngữ lập trình máy tính khác. Đây được gọi là nhiệm vụ dịch mã vì đầu vào là một ngôn ngữ lập trình như Java và đầu ra là một ngôn ngữ lập trình khác như Python.

Hơn nữa, tối ưu hóa mã sẽ cải thiện mã trung gian bằng cách sử dụng các nguyên tắc mã như nguyên tắc SOLID và các cách khác, cũng như bằng cách giảm mức tiêu thụ tài nguyên của nó (tức là CPU, Bộ nhớ). Vì vậy, chúng tôi sẽ tiếp tục dự đoán mã được tối ưu hóa với độ chính xác cao và xem xét các chủ đề lý thuyết mã như nguyên tắc SOLID và khái niệm OOP. Hơn nữa, chúng tôi sẽ cố gắng viết nhiều dòng mã. Hơn nữa, chúng tôi sẽ cố gắng viết nhiều dòng mã và trình bày cách tạo mã nguồn cho một ngôn ngữ lập trình nhất định từ một ngôn ngữ lập trình máy tính khác. Đây được gọi là tác vụ dịch mã vì đầu vào là ngôn ngữ lập trình như Java và đầu ra là ngôn ngữ lập trình khác như Python.

Các hướng nghiên cứu trong tương lai trong lĩnh vực Mô hình ngôn ngữ lớn (LLM) để hỗ trợ mã đưa ra những con đường đầy hứa hẹn để khám phá và phát triển. Một hướng quan trọng liên quan đến việc đi sâu vào lĩnh vực Hỗ trợ mã được cá nhân hóa, bao gồm việc điều chỉnh LLM theo sở thích của từng nhà phát triển, phong cách mã hóa và trình độ kỹ năng để nâng cao hiệu quả của chúng trong việc đáp ứng các nhu cầu và quy trình làm việc riêng biệt của các

nha phat trien.

Một lĩnh vực quan trọng khác cho cuộc điều tra trong tương lai là Tích hợp liên mạch LLM vào Môi trường phát triển phổ biến. Điều này liên quan đến các nỗ lực hướng tới việc đạt được sự tích hợp hài hòa và theo thời gian thực của LLM vào các môi trường phát triển được sử dụng rộng rãi để cung cấp cho các nhà phát triển sự hỗ trợ ngay lập tức và phù hợp với ngữ cảnh trong quá trình viết mã của họ.

Hơn nữa, việc đánh giá các kiến trúc LLM khác nhau nổi lên như một hướng nghiên cứu quan trọng. Đánh giá toàn diện các kiến trúc LLM khác nhau là điều cần thiết để phân biệt điểm mạnh và điểm yếu của chúng trong các tác vụ hỗ trợ mã cụ thể, góp phần tinh chỉnh việc lựa chọn và triển khai LLM dựa trên tính phù hợp của chúng đối với các thách thức lập trình đa dạng.

Ngoài ra, việc khám phá các kỹ thuật AI có thể giải thích được là một hướng nghiên cứu hấp dẫn trong tương lai về LLM để hỗ trợ mã. Việc kết hợp các phương pháp giúp quy trình ra quyết định của LLM trở nên minh bạch hơn có thể cung cấp những hiểu biết sâu sắc hơn cho các nhà phát triển, tăng cường lòng tin của người dùng và góp phần hiểu rõ hơn về cách LLM đóng góp vào quy trình viết mã. Cuối cùng, việc giải quyết Sự thiên vị và Công bằng nổi lên như một khía cạnh quan trọng của các nỗ lực nghiên cứu trong tương lai. Các nỗ lực nên được hướng tới việc xác định và giảm thiểu các thiên vị tiềm ẩn trong LLM để đảm bảo rằng hỗ trợ mã vẫn công bằng và toàn diện cho tất cả các nhà phát triển. Các hướng nghiên cứu này phù hợp với các cân nhắc đạo đức rộng hơn liên quan đến việc triển khai các công nghệ AI trong môi trường lập trình.

References

- [1] A. Soliman, S. Shaheen, and M. Hadhoud, "Leveraging pre-trained language models for code generation," *Complex & Intelligent Systems*, 2024.
- [2] A. S. Soliman, M. M. Hadhoud, and S. I. Shaheen, "Mariancg: A code generation transformer model inspired by machine translation," *Journal of Engineering and Applied Science*, vol. 69, 2022.
- [3] A. Soliman, S. Shaheen, and M. Hadhoud, "Effect of language models on programming assistance capabilities," 2024, Manuscript submitted for publication.
- [4] J. Cheng, S. Reddy, V. Saraswat, and M. Lapata, "Learning an Executable Neural Semantic Parser," *Computational Linguistics*, vol. 45, no. 1, pp. 59–94, Mar. 2019.
- [5] W. Ling *et al.*, "Latent predictor networks for code generation," *ArXiv*, vol. abs/1603.06744, 2016.
- [6] P. Yin and G. Neubig, "A syntactic neural model for general-purpose code generation," *arXiv preprint arXiv:1704.01696*, 2017.
- [7] M. Rabinovich, M. Stern, and D. Klein, "Abstract syntax networks for code generation and semantic parsing," *arXiv preprint arXiv:1704.07535*, 2017.
- [8] P. Joshi, N. Goyal, and M. Choudhury, "Semantic parsing and its applications in natural language programming," 2018.
- [9] A. Kamath and R. Das, "A survey on semantic parsing," *ArXiv*, vol. abs/1812.00978, 2018. [Online]. Available: <https://api.semanticscholar.org/CorpusID:54181011>.
- [10] B. Wang, M. Lapata, and I. Titov, "Learning from executions for semantic parsing," 2021. arXiv: 2104.05819 [cs.CL].
- [11] P. Pasupat, Y. Zhang, and K. Guu, "Controllable semantic parsing via retrieval augmentation," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 7683–7698. [Online]. Available: <https://aclanthology.org/2021.emnlp-main.607>.
- [12] P. Joshi, "Semantic parsing and its applications in natural language programming," 2018. [Online]. Available: <https://pratikmjoshi.github.io/assets/thesis.pdf>.
- [13] Y. Chen and M. Bansal, "Controllable semantic parsing via retrieval augmentation," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, ACL Anthology, 2021, pp. 7582–7592. [Online]. Available: <https://aclanthology.org/2021.emnlp-main.607>.

- [14] Y. Li and M. Jaggi, "A survey on semantic parsing," *OpenReview*, vol. 1, pp. 1–29, 2017. [Online]. Available: <https://openreview.net/pdf?id=HylaEWcTT7>.
- [15] L. Dong and M. Lapata, "Learning an executable neural semantic parser," *Computational Linguistics*, vol. 45, no. 1, pp. 59–106, 2019. [Online]. Available: <https://direct.mit.edu/coli/article/45/1/59/1613/Learning-an-Executable-Neural-Semantic-Parser>.
- [16] Y. Chen and R. J. Mooney, "Learning to interpret natural language commands through human-robot dialog," *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, pp. 1446–1451, 2011.
- [17] J. Krishnamurthy and T. Mitchell, "Weakly supervised training of semantic parsers," *Transactions of the Association for Computational Linguistics*, vol. 1, pp. 49–62, 2012.
- [18] Y. Artzi and L. Zettlemoyer, "Weakly supervised learning of semantic parsers for mapping instructions to actions," *Transactions of the Association for Computational Linguistics*, vol. 1, pp. 49–62, 2013.
- [19] D. Hendrycks *et al.*, "Measuring coding challenge competence with apps," 2021. arXiv: 2105.09938 [cs.SE].
- [20] J. Liu, C. S. Xia, Y. Wang, and L. Zhang, *Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation*, 2023. arXiv: 2305.01210 [cs.SE].
- [21] D. Zan *et al.*, "When neural model meets nl2code: A survey," *ArXiv*, vol. abs/2212.09420, 2022.
- [22] T. H. Le, H. Chen, and M. A. Babar, "Deep learning for source code modeling and generation: Models, applications, and challenges," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–38, 2020.
- [23] M. L. Siddiq, B. Casey, and J. Santos, "A lightweight framework for high-quality code generation," Jul. 2023.
- [24] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, "The impact of ai on developer productivity: Evidence from github copilot," Feb. 2023. [Online]. Available: <http://arxiv.org/abs/2302.06590>.
- [25] P. Barry, *Head First Python, 2nd Edition*. O'Reilly Media, 2017, ch. 4. [Online]. Available: <https://www.oreilly.com/library/view/head-first-python/9781491919521/>.
- [26] Z. Li, Y. Jiang, X. J. Zhang, and H. Y. Xu, "The metric for automatic code generation," *Procedia Computer Science*, vol. 166, pp. 279–286, 2020, Proceedings of the 3rd International Conference on Mechatronics and Intelligent Robotics (ICMIR-2019). [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1877050920302210>.
- [27] S. Ayupov and N. Chirkova, "Parameter-efficient finetuning of transformers for source code," Dec. 2022. [Online]. Available: <http://arxiv.org/abs/2212.05901>.

- [28] F. F. Xu, B. Vasilescu, and G. Neubig, "In-ide code generation from natural language: Promise and challenges," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 2, pp. 1–47, 2022.
- [29] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [30] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.
- [31] T. B. Brown *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.
- [32] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.
- [33] R. Shin *et al.*, "Constrained language models yield few-shot semantic parsers," *arXiv preprint arXiv:2104.08768*, 2021.
- [34] T. Adhikari, "Investigating the use of natural language processing for automated code generation," *SSRN Electronic Journal*, Apr. 2023.
- [35] M. Salvagno, ChatGPT, F. Taccone, and A. G. Gerli, "Can artificial intelligence help for scientific writing?" *Critical care (London, England)*, vol. 27, p. 75, Feb. 2023.
- [36] A. Torfi, R. A. Shirvani, Y. Keneshloo, N. Tavaf, and E. A. Fox, "Natural language processing advancements by deep learning: A survey," Mar. 2020. [Online]. Available: <http://arxiv.org/abs/2003.01200>.
- [37] L. Alzubaidi *et al.*, "Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions," *Journal of Big Data*, vol. 8, 2021. [Online]. Available: <https://api.semanticscholar.org/CorpusID:232434552>.
- [38] C. C. Aggarwal, *Neural Networks and Deep Learning*. Springer Cham, 2018.
- [39] I. H. Sarker, "Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions," *Sn Computer Science*, vol. 2, 2021. [Online]. Available: <https://api.semanticscholar.org/CorpusID:237241776>.
- [40] J. Patterson and A. Gibson, *Major Architectures of Deep Networks*. O'Reilly Media, Inc., 2017, ch. 4. [Online]. Available: <https://www.oreilly.com/library/view/deep-learning/9781491924570/>.
- [41] "Artificial intelligence: A powerful paradigm for scientific research," *Innovation*, vol. 2, 4 Nov. 2021.
- [42] I. H. Sarker, "Machine learning: Algorithms, real-world applications and research directions," *SN Computer Science*, vol. 2, 3 May 2021.
- [43] A. Gillioz, J. Casas, E. Mugellini, and O. A. Khaled, "Overview of the transformer-based models for nlp tasks," *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*, pp. 179–183, 2020. [Online]. Available: <https://api.semanticscholar.org/CorpusID:222296346>.

- [44] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," Aug. 2017. [Online]. Available: <http://arxiv.org/abs/1708.02709>.
- [45] D. Khurana, A. Koli, K. Khatter, and S. Singh, "Natural language processing: State of the art, current trends and challenges," [Online]. Available: <https://doi.org/10.1007/s11042-022-13428-4>.
- [46] "Beyond the hype: Big data concepts, methods, and analytics," *International Journal of Information Management*, vol. 35, pp. 137–144, 2 2015.
- [47] M. Wankhade, A. C. S. Rao, and C. Kulkarni, "A survey on sentiment analysis methods, applications, and challenges," *Artificial Intelligence Review*, vol. 55, pp. 5731–5780, 7 Oct. 2022.
- [48] X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang, "Pre-trained models for natural language processing: A survey," Mar. 2020. [Online]. Available: <http://arxiv.org/abs/2003.08271> <https://doi.org/10.1007/s11431-020-1647-3>.
- [49] U. NASEEM, I. RAZZAK, S. K. KHAN, and M. PRASAD, "A comprehensive survey on word representation models: From classical to state-of-the-art word representation language models," *arxiv*, vol. 5, CSCW1 Apr. 2021.
- [50] P. Cerda, G. Varoquaux, and B. Kégl, "Similarity encoding for learning with dirty categorical variables," *Machine Learning*, vol. 107, pp. 1477–1494, 2018. [Online]. Available: <https://api.semanticscholar.org/CorpusID:46927512>.
- [51] I. Banerjee, S. Madhavan, R. E. Goldman, and D. L. Rubin, "Intelligent word embeddings of free-text radiology reports," [Online]. Available: <https://github.com/imonban/RadiologyReportEmbedding>.
- [52] S. Hasni and Â. S. Faiz, "Word embeddings and deep learning for location prediction: Tracking coronavirus from british and american tweets," *Social Network Analysis and Mining*, vol. 11, p. 66, 2021. [Online]. Available: <https://doi.org/10.1007/s13278-021-00777-5>.
- [53] Y. Goldberg and O. Levy, "Word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method," *ArXiv*, vol. abs/1402.3722, 2014.
- [54] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," 2014. [Online]. Available: <https://api.semanticscholar.org/CorpusID:1957433>.
- [55] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "Fasttext.zip: Compressing text classification models," *ArXiv*, vol. abs/1612.03651, 2016.
- [56] M. Apidianaki, "From word types to tokens and back: A survey of approaches to word meaning representation and interpretation," 2023. [Online]. Available: <https://doi.org/10.1162/coli>.
- [57] E. Sezerer and S. Tekir, "A survey on neural word embeddings," Oct. 2021. [Online]. Available: <http://arxiv.org/abs/2110.01804>.
- [58] M. E. Peters *et al.*, "Deep contextualized word representations," *ArXiv*, vol. abs/1802.05365, 2018.

- [59] A. Radford and K. Narasimhan, "Improving language understanding by generative pre-training," 2018.
- [60] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *ArXiv*, vol. abs/1810.04805, 2019.
- [61] K. Ethayarajh, "How contextual are contextualized word representations? comparing the geometry of bert, elmo, and gpt-2 embeddings," 2019. [Online]. Available: <https://api.semanticscholar.org/CorpusID:202120592>.
- [62] H. E. Boukkouri, O. Ferret, T. Lavergne, H. Noji, P. Zweigenbaum, and J. Tsujii, "Characterbert: Reconciling elmo and bert for word-level open-vocabulary representations from characters," 2020. [Online]. Available: <https://api.semanticscholar.org/CorpusID:224803588>.
- [63] A. Goldstein *et al.*, "Thinking ahead: Spontaneous next word predictions in context as a keystone of language in humans and machines," [Online]. Available: <https://doi.org/10.1101/2020.12.02.403477>.
- [64] K. Bostrom and G. Durrett, "Byte pair encoding is suboptimal for language model pretraining," 2020. [Online]. Available: <https://api.semanticscholar.org/CorpusID:215416175>.
- [65] N. Casas, M. R. Costa-jussà, and J. A. R. Fonollosa, "Combining subword representations into word-level representations in the transformer architecture," 2020. [Online]. Available: <https://api.semanticscholar.org/CorpusID:220058122>.
- [66] X. Han *et al.*, "Pre-trained models: Past, present and future," *AI Open*, vol. 2, pp. 225–250, 2021. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2666651021000231>.
- [67] K. S. Kalyan, A. Rajasekharan, and S. Sangeetha, "Ammus : A survey of transformer-based pretrained models in natural language processing," *ArXiv*, vol. abs/2108.05542, 2021.
- [68] N. Patwardhan, S. Marrone, and C. Sansone, *Transformers in the real world: A survey on nlp applications*, Apr. 2023.
- [69] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning in natural language processing," Jul. 2018. [Online]. Available: <http://arxiv.org/abs/1807.10854>.
- [70] P. KÅosowski, "Deep learning for natural language processing and language modelling," *2018 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*, pp. 223–228, 2018.
- [71] N. Soni, M. Matero, N. Balasubramanian, and H. A. Schwartz, "Human language modeling," in *FINDINGS*, 2022.
- [72] J. Pilault, R. Li, S. Subramanian, and C. J. Pal, "On extractive and abstractive neural document summarization with transformer language models," *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 9308–9319, 2020. [Online]. Available: <https://api.semanticscholar.org/CorpusID:202541012>.

- [73] D. F. Coimbra, "Hybrid extractive/abstractive summarization using pre-trained sequence-to-sequence models," 2020. [Online]. Available: <https://api.semanticscholar.org/CorpusID:244114480>.
- [74] M. P. Karnik and D. V. Kodavade, "Abstractive summarization with efficient transformer based approach," *International Journal on Recent and Innovation Trends in Computing and Communication*, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:259676532>.
- [75] H. Wang, H. Wu, Z. He, L. Huang, and K. W. Church, "Progress in machine translation," *Engineering*, 2021. [Online]. Available: <https://api.semanticscholar.org/CorpusID:237767709>.
- [76] V. Mavi, A. Jangra, and A. Jatowt, "A survey on multi-hop question answering and generation," *ArXiv*, vol. abs/2204.09140, 2022.
- [77] M. Li *et al.*, "Trocr: Transformer-based optical character recognition with pre-trained models," 2021. [Online]. Available: <https://api.semanticscholar.org/CorpusID:237581568>.
- [78] D. Khurana, A. Koli, K. Khatter, and S. Singh, "Natural language processing: State of the art, current trends and challenges," *Multimedia Tools and Applications*, vol. 82, pp. 3713–3744, 2017. [Online]. Available: <https://api.semanticscholar.org/CorpusID:7678100>.
- [79] D. Jurafsky and J. Martin., *Speech and Language Processing*. Stanford University, 2023, ch. 3.
- [80] M. Suzuki, N. Itoh, T. Nagano, G. Kurata, and S. Thomas, "Improvements to n-gram language model using text generated from neural language model," *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 7245–7249, 2019. [Online]. Available: <https://api.semanticscholar.org/CorpusID:146074612>.
- [81] H. Wang, J. He, X. Zhang, and S. Liu, "A short text classification method based on n gram and cnn," *Chinese Journal of Electronics*, vol. 29, pp. 248–254, 2020. [Online]. Available: <https://api.semanticscholar.org/CorpusID:216254591>.
- [82] M. Tang, K. Wang, and S. Pajwani, "Lecture 12: Data types i: Sets, text, n-grams, nlp," 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:247634773>.
- [83] I. Hamed, M. S. Elmahdy, and S. Abdennadher, "Expanding n-grams for code-switch language models," 2018. [Online]. Available: <https://api.semanticscholar.org/CorpusID:52135655>.
- [84] A. K. Lampinen *et al.*, "Can language models learn from explanations in context?" In *Conference on Empirical Methods in Natural Language Processing*, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:247957917>.
- [85] F. Liu, G. Li, Y. Zhao, and Z. Jin, "Multi-task learning based pre-trained language model for code completion," *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 473–485, 2020. [Online]. Available: <https://api.semanticscholar.org/CorpusID:229703606>.

- [86] E. L. Dolson, "Calculating lexicase selection probabilities is np-hard," *Proceedings of the Genetic and Evolutionary Computation Conference*, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:255942492>.
- [87] F. Mezzoudj and A. Benyettou, "An empirical study of statistical language models: N-gram language models vs. neural network language models," *International Journal of Innovative Computing and Applications*, vol. 9, p. 189, 2018.
- [88] M. Levit *et al.*, "Personalization of word-phrase-entity language models," in *Interspeech*, 2015.
- [89] P. Linardatos, V. Papastefanopoulos, and S. B. Kotsiantis, "Explainable ai: A review of machine learning interpretability methods," *Entropy*, vol. 23, 2020. [Online]. Available: <https://api.semanticscholar.org/CorpusID:229722844>.
- [90] M. R. Islam, M. U. Ahmed, S. Barua, and S. Begum, "A systematic review of explainable artificial intelligence in terms of different application domains and tasks," *Applied Sciences*, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:246393748>.
- [91] M. Labeau, "Neural language models : Dealing with large vocabularies," 2018. [Online]. Available: <https://api.semanticscholar.org/CorpusID:86639203>.
- [92] T. Luong, M. Kayser, and C. D. Manning, "Deep neural language models for machine translation," in *Conference on Computational Natural Language Learning*, 2015. [Online]. Available: <https://api.semanticscholar.org/CorpusID:6850604>.
- [93] S. Yang, Y. Wang, and X. Chu, "A survey of deep learning techniques for neural machine translation," *ArXiv*, vol. abs/2002.07526, 2020.
- [94] Z. Zhu, H. Yu, C. Shen, J. Du, Z. Shen, and Z. Wang, "Causal language model aided sequential decoding with natural redundancy," *IEEE Transactions on Communications*, vol. 71, pp. 2685–2697, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:257147253>.
- [95] H. Liu *et al.*, "Towards better few-shot and finetuning performance with forgetful causal language models," 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:256416540>.
- [96] R. M. Schmidt, "Recurrent neural networks (rnns): A gentle introduction and overview," *ArXiv*, vol. abs/1912.05911, 2019.
- [97] P. Baheti. "The essential guide to neural network architectures." (2021), [Online]. Available: <https://www.v7labs.com/blog/neural-network-architectures-guide> (visited on 08/11/2023).
- [98] R. DiPietro and G. D. Hager, "Deep learning: Rnns and lstm," *Handbook of Medical Image Computing and Computer Assisted Intervention*, pp. 503–519, Jan. 2020.

- [99] R. C. Staudemeyer and E. R. Morris, "Understanding lstm - a tutorial into long short-term memory recurrent neural networks," *ArXiv*, vol. abs/1909.09586, 2019. [Online]. Available: <https://api.semanticscholar.org/CorpusID:202712597>.
- [100] S. Yang, X. Yu, and Y. Zhou, "Lstm and gru neural network performance comparison study: Taking yelp review dataset as an example," *2020 International Workshop on Electronic Communication and Artificial Intelligence (IWECAI)*, pp. 98–101, 2020. [Online]. Available: <https://api.semanticscholar.org/CorpusID:222418540>.
- [101] J. Duan, P. Zhang, R. Qiu, and Z.-L. Huang, "Long short-term enhanced memory for sequential recommendation," *World Wide Web*, vol. 26, pp. 561–583, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:248788714>.
- [102] Y. Liu *et al.*, "Roberta: A robustly optimized bert pretraining approach," *ArXiv*, vol. abs/1907.11692, 2019.
- [103] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A survey of transformers," *AI Open*, vol. 3, pp. 111–132, 2021. [Online]. Available: <https://api.semanticscholar.org/CorpusID:235368340>.
- [104] H. Lu, L. Ehwerhemuepha, and C. Rakovski, "A comparative study on deep learning models for text classification of unstructured medical notes with various levels of class imbalance," *BMC Medical Research Methodology*, vol. 22, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:250181475>.
- [105] C. Raffel *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer," *ArXiv*, vol. abs/1910.10683, 2020.
- [106] L. Ouyang *et al.*, "Training language models to follow instructions with human feedback," *ArXiv*, vol. abs/2203.02155, 2022.
- [107] J. Wei *et al.*, "Chain of thought prompting elicits reasoning in large language models," *ArXiv*, vol. abs/2201.11903, 2022.
- [108] M. Zhang, G. Geng, and J. Chen, "Semi-supervised bidirectional long short-term memory and conditional random fields model for named-entity recognition using embeddings from language models representations," *Entropy*, vol. 22, p. 252, Feb. 2020.
- [109] A. Rahali and M. A. Akhloufi, "End-to-end transformer-based models in textual-based nlp," *AI*, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:255707282>.
- [110] C. Raffel *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer," *ArXiv*, vol. abs/1910.10683, 2019.
- [111] H.-Y. Zhou *et al.*, "A transformer-based representation-learning model with unified processing of multimodal input for clinical diagnostics," *Nature Biomedical Engineering*, vol. 7, pp. 743–755, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:258999463>.

- [112] N. H. Thanh *et al.*, "Transformer-based approaches for legal text processing," *The Review of Socionetwork Strategies*, pp. 1–21, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:246281564>.
- [113] X. Dai, I. Chalkidis, S. Darkner, and D. Elliott, "Revisiting transformer-based models for long document classification," in *Conference on Empirical Methods in Natural Language Processing*, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:248177894>.
- [114] Q. Zheng *et al.*, "Codegeex: A pre-trained model for code generation with multilingual evaluations on humaneval-x," *ArXiv*, vol. abs/2303.17568, 2023.
- [115] X. Amatriain, "Transformer models: An introduction and catalog," *ArXiv*, vol. abs/2302.07730, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:256868804>.
- [116] T. Schick *et al.*, "Toolformer: Language models can teach themselves to use tools," *ArXiv*, vol. abs/2302.04761, 2023.
- [117] S. I. Ross, F. Martinez, S. Houde, M. J. Muller, and J. D. Weisz, "The programmer's assistant: Conversational interaction with a large language model for software development," *ArXiv*, vol. abs/2302.07080, 2023.
- [118] I. Solaiman, "The gradient of generative ai release: Methods and considerations," *ArXiv*, vol. abs/2302.04844, 2023.
- [119] S. Frieder *et al.*, "Mathematical capabilities of chatgpt," *ArXiv*, vol. abs/2301.13867, 2023.
- [120] G. Todd, S. Earle, M. U. Nasir, M. C. Green, and J. Togelius, "Level generation through large language models," *ArXiv*, vol. abs/2302.05817, 2023.
- [121] J. Koco'n *et al.*, "Chatgpt: Jack of all trades, master of none," *ArXiv*, vol. abs/2302.10724, 2023.
- [122] M. E. Peters *et al.*, "Deep contextualized word representations," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp. 2227–2237. [Online]. Available: <https://aclanthology.org/N18-1202>.
- [123] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.
- [124] Y. Gu, X. Han, Z. Liu, and M. Huang, "Ppt: Pre-trained prompt tuning for few-shot learning," in *ACL*, 2022.
- [125] N. Ding *et al.*, "Delta tuning: A comprehensive study of parameter efficient methods for pre-trained language models," *ArXiv*, vol. abs/2203.06904, 2022.
- [126] Y. Qin *et al.*, "Elle: Efficient lifelong pre-training for emerging data," in *FINDINGS*, 2022.
- [127] L. Alzubaidi *et al.*, "Towards a better understanding of transfer learning for medical imaging: A case study," *Applied Sciences*, 2020. [Online]. Available: <https://api.semanticscholar.org/CorpusID:225769039>.

- [128] G. E. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *ArXiv*, vol. abs/1503.02531, 2015.
- [129] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, pp. 1789–1819, 2020.
- [130] C.-Y. Hsieh *et al.*, "Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes," May 2023. [Online]. Available: <http://arxiv.org/abs/2305.02301>.
- [131] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: A method for automatic evaluation of machine translation," in *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.
- [132] M. Ashfaque and S. A. Hannan, "Study and evaluation of "seq-2-seq" model competency in ai-based educational chatbot for the marathi language," vol. 12, pp. 223–232, Aug. 2023.
- [133] M. Post, "A call for clarity in reporting bleu scores," *arXiv preprint arXiv:1804.08771*, 2018.
- [134] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*, Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: <https://aclanthology.org/W04-1013>.
- [135] B. Loriot, F. Madeiral, and M. Martin, "Styler: Learning formatting conventions to repair checkstyle violations," *Empirical Software Engineering*, vol. 27, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:254470409>.
- [136] R. Hart, B. Hays, C. McMillin, E. K. Rezig, G. Rodriguez-Rivera, and J. A. Turkstra, "Eastwood-tidy: C linting for automated code style assessment in programming courses," *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:257311662>.
- [137] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "Unixeroder: Unified cross-modal pre-training for code representation," in *Annual Meeting of the Association for Computational Linguistics*, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:247315559>.
- [138] N. D. C. Oliveira, M. Ribeiro, R. Bonifácio, R. Gheyi, I. S. Wiese, and B. F. dos Santos Neto, "Lint-based warnings in python code: Frequency, awareness and refactoring," *2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pp. 208–218, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:255778017>.
- [139] D. A. Kapustin, V. V. Shvyrov, E. Y. Suvorova, and T. I. Shulyka, "Pep 8 compliance analysis for python open source projects," *Programmnaya Ingeneriya*, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:256184826>.
- [140] Y. Ryou, S. Joh, J. Yang, S. Kim, and Y. Kim, "Code understanding linter to detect variable misuse," *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:255441367>.

- [141] O. Karnalim, Simon, and W. J. Chivers, "Work-in-progress: Code quality issues of computing undergraduates," *2022 IEEE Global Engineering Education Conference (EDUCON)*, pp. 1734–1736, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:248699287>.
- [142] G. Wang, J. Chen, J. Gao, and Z. Huang, "Python code smell refactoring route generation based on association rule and correlation," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 31, pp. 1329–1347, 2021. [Online]. Available: <https://api.semanticscholar.org/CorpusID:244201485>.
- [143] E. Dehaerne, B. Dey, S. Halder, S. Gendt, and W. Meert, "Code generation using machine learning: A systematic review," *IEEE Access*, vol. 10, pp. 1–1, Jan. 2022.
- [144] B. Wang, M. Lapata, and I. Titov, "Learning from executions for semantic parsing," *ArXiv*, vol. abs/2104.05819, 2021. [Online]. Available: <https://api.semanticscholar.org/CorpusID:233219560>.
- [145] C. Yang, Y. Liu, and C. Yin, "Recent advances in intelligent source code generation: A survey on natural language based studies," *Entropy*, vol. 23, 2021. [Online]. Available: <https://api.semanticscholar.org/CorpusID:237938627>.
- [146] H. Ye, W. Li, and L. Wang, "Jointly learning semantic parser and natural language generator via dual information maximization," *ArXiv*, vol. abs/1906.00575, 2019. [Online]. Available: <https://api.semanticscholar.org/CorpusID:173990567>.
- [147] H. Dittmer, "Code generation based on controlled natural language input," *Software Engineering Advances*, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:259980304>.
- [148] H. Sellik, "Natural language processing techniques for code generation," 2019. [Online]. Available: <https://api.semanticscholar.org/CorpusID:214088887>.
- [149] C. Dong *et al.*, "A survey of natural language generation," *ACM Computing Surveys*, vol. 55, pp. 1–38, 8 Aug. 2023.
- [150] E. Al-Hossami and S. Shaikh, "A survey on artificial intelligence for source code: A dialogue systems perspective," Feb. 2022. [Online]. Available: <http://arxiv.org/abs/2202.04847>.
- [151] F. F. Xu, U. Alon, G. Neubig, and V. J. Hellendoorn, "A systematic evaluation of large language models of code," Feb. 2022. [Online]. Available: <http://arxiv.org/abs/2202.13169>.
- [152] "Natural language to code using transformers," Feb. 2022. [Online]. Available: <http://arxiv.org/abs/2202.00367>.
- [153] P. Yin *et al.*, "Natural language to code generation in interactive data science notebooks," Dec. 2022. [Online]. Available: <http://arxiv.org/abs/2212.09248>.
- [154] J. Gu, Z. Lu, H. Li, and V. O. K. Li, "Incorporating copying mechanism in sequence-to-sequence learning," *ArXiv*, vol. abs/1603.06393, 2016. [Online]. Available: <https://api.semanticscholar.org/CorpusID:8174613>.

- [155] S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, “Mapping language to code in programmatic context,” in *Conference on Empirical Methods in Natural Language Processing*, 2018. [Online]. Available: <https://api.semanticscholar.org/CorpusID:52125417>.
- [156] C. Xiao, M. Dymetman, and C. Gardent, “Sequence-based structured prediction for semantic parsing,” in *Annual Meeting of the Association for Computational Linguistics*, 2016. [Online]. Available: <https://api.semanticscholar.org/CorpusID:16911296>.
- [157] J. Krishnamurthy, P. Dasigi, and M. Gardner, “Neural semantic parsing with type constraints for semi-structured tables,” in *Conference on Empirical Methods in Natural Language Processing*, 2017. [Online]. Available: <https://api.semanticscholar.org/CorpusID:1675452>.
- [158] S. Iyer, A. Cheung, and L. Zettlemoyer, “Learning programmatic idioms for scalable semantic parsing,” in *Conference on Empirical Methods in Natural Language Processing*, 2019. [Online]. Available: <https://api.semanticscholar.org/CorpusID:125969731>.
- [159] E. C. Shin, M. Allamanis, M. Brockschmidt, and A. Polozov, “Program synthesis and semantic parsing with learned code idioms,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- [160] M. Nye, L. B. Hewitt, J. B. Tenenbaum, and A. Solar-Lezama, “Learning to infer program sketches,” *ArXiv*, vol. abs/1902.06349, 2019.
- [161] L. Dong, C. Quirk, and M. Lapata, “Confidence modeling for neural semantic parsing,” in *Annual Meeting of the Association for Computational Linguistics*, 2018. [Online]. Available: <https://api.semanticscholar.org/CorpusID:13686145>.
- [162] S. Chaurasia and R. J. Mooney, “Dialog for language to code,” in *International Joint Conference on Natural Language Processing*, 2017. [Online]. Available: <https://api.semanticscholar.org/CorpusID:217279086>.
- [163] J. Andreas *et al.*, “Task-oriented dialogue as dataflow synthesis,” *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 556–571, 2020. [Online]. Available: <https://api.semanticscholar.org/CorpusID:221822514>.
- [164] O. Polozov and S. Gulwani, “Flashmeta: A framework for inductive program synthesis,” *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2015.
- [165] E. Parisotto, A.-r. Mohamed, R. Singh, L. Li, D. Zhou, and P. Kohli, “Neuro-symbolic program synthesis,” *ArXiv*, vol. abs/1611.01855, 2017.
- [166] S. Bhupatiraju, R. Singh, A.-r. Mohamed, and P. Kohli, “Deep api programmer: Learning to program with apis,” *ArXiv*, vol. abs/1704.04327, 2017.
- [167] M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow, “Deepcoder: Learning to write programs,” *ArXiv*, vol. abs/1611.01989, 2017.

- [168] J. Devlin, J. Uesato, S. Bhupatiraju, R. Singh, A.-r. Mohamed, and P. Kohli, “Robustfill: Neural program learning under noisy i/o,” *ArXiv*, vol. abs/1703.07469, 2017.
- [169] Y. Xu, L. Dai, U. Singh, K. Zhang, and Z. Tu, “Neural program synthesis by self-learning,” *ArXiv*, vol. abs/1910.05865, 2019.
- [170] I. Polosukhin and A. Skidanov, “Neural program search: Solving data processing tasks from description and examples,” in *ICLR 2018*, 2018.
- [171] L. Dong and M. Lapata, “Language to logical form with neural attention,” *arXiv preprint arXiv:1601.01280*, 2016.
- [172] S. Z. Tongliang Li and Z. Li, “Sp-nlg: A semantic-parsing-guided natural language generation framework,” *Electronics*, 2023. [Online]. Available: <https://doi.org/10.3390/electronics12081772>.
- [173] S. Dahal, A. Maharana, and M. Bansal, “Analysis of tree-structured architectures for code generation,” in *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, 2021, pp. 4382–4391.
- [174] P. Qin, W. Tan, J. Guo, B. Shen, and Q. Tang, “Achieving semantic consistency for multilingual sentence representation using an explainable machine natural language parser (mparser),” *Applied Sciences*, 2021. [Online]. Available: <https://api.semanticscholar.org/CorpusID:245080148>.
- [175] P. Yin and G. Neubig, “Tranx: A transition-based neural abstract syntax parser for semantic parsing and code generation,” *arXiv preprint arXiv:1810.02720*, 2018.
- [176] P. Yin and G. Neubig, “Reranking for neural semantic parsing,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.
- [177] Z. Sun, Q. Zhu, Y. Xiong, Y. Sun, L. Mou, and L. Zhang, “Treegen: A tree-based transformer architecture for code generation,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 8984–8991.
- [178] F. F. Xu, Z. Jiang, P. Yin, B. Vasilescu, and G. Neubig, “Incorporating external knowledge through pre-training for natural language to code generation,” *arXiv preprint arXiv:2004.09015*, 2020.
- [179] K. Lano and Q. Xue, “Code generation by example using symbolic machine learning,” *SN Computer Science*, vol. 4, pp. 1–23, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:256158586>.
- [180] Z. Tang *et al.*, “Ast-trans: Code summarization with efficient tree-structured attention,” 2022 *IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, pp. 150–162, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:249305308>.
- [181] T. H. M. Le, H. Chen, and M. A. Babar, “Deep learning for source code modeling and generation,” *ACM Computing Surveys (CSUR)*, vol. 53, pp. 1–38, 2020. [Online]. Available: <https://api.semanticscholar.org/CorpusID:211096967>.

- [182] S. Norouzi, K. Tang, and Y. Cao, “Code generation from natural language with less prior knowledge and more monolingual data,” in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, 2021, pp. 776–785.
- [183] G. Orlanski and A. Gittens, “Reading stackoverflow encourages cheating: Adding question text improves extractive code generation,” *arXiv preprint arXiv:2106.04447*, 2021.
- [184] N. Beau and B. Crabbé, “The impact of lexical and grammatical processing on generating code from natural language,” *arXiv preprint arXiv:2202.13972*, 2022.
- [185] Z. Wang, G. Cuenca, S. Zhou, F. F. Xu, and G. Neubig, “Mconala: A benchmark for code generation from multiple natural languages,” *arXiv preprint arXiv:2203.08388*, 2022.
- [186] U. Kusupati and V. R. T. Ailavarapu, “Natural language to code using transformers,” *ArXiv*, vol. abs/2202.00367, 2022.
- [187] E. Al-Hossami and S. Shaikh, “A survey on artificial intelligence for source code: A dialogue systems perspective,” *ArXiv*, vol. abs/2202.04847, 2022.
- [188] P. Ni, R. Okhrati, S. Guan, and V. I. Chang, “Knowledge graph and deep learning-based text-to-graphql model for intelligent medical consultation chatbot,” *Information Systems Frontiers*, pp. 1–20, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:250335068>.
- [189] P. Yin, B. Deng, E. Chen, B. Vasilescu, and G. Neubig, “Learning to mine aligned code and natural language pairs from stack overflow,” in *2018 IEEE/ACM 15th international conference on mining software repositories (MSR)*, IEEE, 2018, pp. 476–486.
- [190] K. Cho *et al.*, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, 2014.
- [191] M. Lewis *et al.*, “Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension,” *arXiv preprint arXiv:1910.13461*, 2019.
- [192] Y. Oda *et al.*, “Learning to generate pseudo-code from source code using statistical machine translation,” in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2015, pp. 574–584.
- [193] H. Husain, H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, “Code-searchnet challenge: Evaluating the state of semantic code search,” *ArXiv*, vol. abs/1909.09436, 2019. [Online]. Available: <https://api.semanticscholar.org/CorpusID:202712680>.
- [194] M. Junczys-Dowmunt *et al.*, “Marian: Fast neural machine translation in c++,” in *Annual Meeting of the Association for Computational Linguistics*, 2018. [Online]. Available: <https://api.semanticscholar.org/CorpusID:4623739>.

- [195] S. Rothe, S. Narayan, and A. Severyn, “Leveraging pre-trained checkpoints for sequence generation tasks,” *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 264–280, 2020.
- [196] T. Kudo and J. Richardson, “Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing,” *arXiv preprint arXiv:1808.06226*, 2018.
- [197] T. Kudo, “Subword regularization: Improving neural network translation models with multiple subword candidates,” *arXiv preprint arXiv:1804.10959*, 2018.
- [198] R. Sennrich, B. Haddow, and A. Birch, “Neural machine translation of rare words with subword units,” *arXiv preprint arXiv:1508.07909*, 2015.
- [199] J. Alammar. “The illustrated transformer.” (2018), [Online]. Available: <http://jalammar.github.io/illustrated-transformer/> (visited on 08/11/2023).
- [200] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter,” *ArXiv*, vol. abs/1910.01108, 2019.
- [201] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, “Electra: Pre-training text encoders as discriminators rather than generators,” *ArXiv*, vol. abs/2003.10555, 2020.
- [202] I. Yamada, A. Asai, H. Shindo, H. Takeda, and Y. Matsumoto, “Luke: Deep contextualized entity representations with entity-aware self-attention,” in *EMNLP*, 2020.
- [203] H. Wang, J. Li, H. Wu, E. Hovy, and Y. Sun, “Pre-trained language models and their applications,” *Engineering*, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2095809922006324>.
- [204] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2017. arXiv: 1412.6980 [cs.LG].
- [205] S. I. Ross, F. Martinez, S. Houde, M. Muller, and J. D. Weisz, “The programmer’s assistant: Conversational interaction with a large language model for software development,” Feb. 2023. [Online]. Available: <http://arxiv.org/abs/2302.07080%20http://dx.doi.org/10.1145/3581641.3584037>.
- [206] R. A. Poldrack, T. Lu, and G. BeguÅ, “Ai-assisted coding: Experiments with gpt-4,” Apr. 2023. [Online]. Available: <http://arxiv.org/abs/2304.13187>.
- [207] E. J. Hu *et al.*, “Lora: Low-rank adaptation of large language models,” *ArXiv*, vol. abs/2106.09685, 2021. [Online]. Available: <https://api.semanticscholar.org/CorpusID:235458009>.
- [208] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, “Qlora: Efficient finetuning of quantized llms,” *ArXiv*, vol. abs/2305.14314, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:258841328>.
- [209] N. Ding *et al.*, “Parameter-efficient fine-tuning of large-scale pre-trained language models,” *Nature Machine Intelligence*, vol. 5, pp. 220–235, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:257316425>.
- [210] Z. Zhang *et al.*, “Unifying the perspectives of nlp and software engineering: A survey on language models for code,” 2023. arXiv: 2311.07989 [cs.CL].

- [211] H. Touvron *et al.*, “Llama 2: Open foundation and fine-tuned chat models,” *ArXiv*, vol. abs/2307.09288, Jul. 2023.
- [212] V. Lalin, V. Deshpande, and A. Rumshisky, “Scaling down to scale up: A guide to parameter-efficient fine-tuning,” 2023. arXiv: 2303.15647 [cs.CL].
- [213] T. Dettmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, “Llm.int8(): 8-bit matrix multiplication for transformers at scale,” in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, Curran Associates, Inc., 2022, pp. 30 318–30 332. [Online]. Available: [https://proceedings.neurips.cc/paper_files/paper/2022/file/c3ba4962c05c49636d4c6206a97e9c8a - Paper-Conference.pdf](https://proceedings.neurips.cc/paper_files/paper/2022/file/c3ba4962c05c49636d4c6206a97e9c8a-Paper-Conference.pdf).
- [214] Lightning AI. “Lora: Ll-m / high speed & low latency transformer language model.” (2023), [Online]. Available: <https://lightning.ai/pages/community/tutorial/lora-llm/>.
- [215] Lightning AI. “Understanding llama adapters.” (2023), [Online]. Available: <https://lightning.ai/pages/community/article/understanding-llama-adapters/>.
- [216] X. L. Li and P. Liang, “Prefix-tuning: Optimizing continuous prompts for generation,” 2021. arXiv: 2101.00190 [cs.CL].
- [217] N. Houlsby *et al.*, “Parameter-efficient transfer learning for nlp,” 2019. arXiv: 1902.00751 [cs.LG].
- [218] R. Zhang *et al.*, “Llama-adapter: Efficient fine-tuning of language models with zero-init attention,” 2023. arXiv: 2303.16199 [cs.CV].

Appendix A

Training and Testing Notebooks

A.1 GitHub Repositories

- MarianCG Model
<https://github.com/AhmedSSoliman/MarianCG-NL-to-Code>
- Hybrid Models
<https://github.com/AhmedSSoliman/Leveraging-Pretrained-Language-Models-for-Code-Generation>
- Llama2-CodeGen
<https://github.com/AhmedSSoliman/Llama2-CodeGen-Fine-Tuning-LLama-2>

A.2 Notebooks

- Training MarianCG on CoNaLa
<https://colab.research.google.com/drive/1Qu9q-SDqQ0zM-q8rp3pLDcAW91xSXmum?usp=sharing>
- Testing MarianCG on CoNaLa
<https://colab.research.google.com/drive/1jCVIV7yBUKrPew-1uYoy1ve7AilflB1G?usp=sharing>
- Training on DJANGO

<https://colab.research.google.com/drive/1vPcfUJb84wLM2S00fY9iWEu0VMn6mYdZ?usp=sharing>

- Testing on DJANGO

<https://colab.research.google.com/drive/1OzQColozdmbrPUw627ZdrxEzEGeRy8eU?usp=sharing>

- CodeGeneration App

https://colab.research.google.com/drive/1OdcX-vlc2_kA9bwYDTYiH_7NQwRpzluA?usp=sharing

- Linting and Correction

https://colab.research.google.com/drive/16FTBY3ci04LGVRT-wwCpMRr9kR4dBc_6?usp=sharing

- Llama2-CodeGen

<https://colab.research.google.com/drive/18sAFC7msV0gJ24wn5gl41nU0QRynfLqG?usp=sharing>

Appendix B

Models and Datasets

B.1 Models

Models are available now at the huggingface hub, and can be used or tested and integrated into any project. You can find these models and easily deal with various datasets through the following links:

- MarianCG-CoNaLa
<https://huggingface.co/AhmedSSoliman/MarianCG-CoNaLa>
- MarianCG-DJANGO
<https://huggingface.co/AhmedSSoliman/MarianCG-DJANGO>
- MarianCG-CoNaLa-Large <https://huggingface.co/AhmedSSoliman/Maria>
nCG-CoNaLa-Large
- RoBERTaMarian
<https://www.huggingface.co/AhmedSSoliman/DistilRoBERTa-Marian-Model-on-CoNaLa>
<https://www.huggingface.co/AhmedSSoliman/DistilRoBERTa-Marian-Model-on-DJANGO>
- BERTMarian
<https://www.huggingface.co/AhmedSSoliman/DistilBERT-Marian-Model-on-CoNaLa>

<https://www.huggingface.co/AhmedSSoliman/DistilBERT-Marian-Model-on-DJANGO>

- ELECTRA Marian

<https://www.huggingface.co/AhmedSSoliman/ELECTRA-Marian-Model-on-CoNaLa>

<https://www.huggingface.co/AhmedSSoliman/ELECTRA-Marian-Model-on-DJANGO>

- LUKE Marian

<https://www.huggingface.co/AhmedSSoliman/LUKE-Marian-Model-on-CoNaLa>

<https://www.huggingface.co/AhmedSSoliman/LUKE-Marian-Model-on-DJANGO>

- Llama2-CodeGen

<https://huggingface.co/AhmedSSoliman/Llama2-CodeGen-PEFT-QLoRA>

B.2 Datasets

- CoNaLa Dataset

<https://huggingface.co/datasets/AhmedSSoliman/CoNaLa-Large>

- DJANGO Dataset

<https://huggingface.co/datasets/AhmedSSoliman/DJANGO>

- CodeSearchNet Dataset

<https://huggingface.co/datasets/AhmedSSoliman/CodeSearchNet-Python>

<https://huggingface.co/datasets/AhmedSSoliman/CodeSearchNet>

الملخص

عملية إنشاء الكود البرمجي تقع كتلةً أساسية في التطوير المعاصر للبرمجيات، حيث تشمل إنشاء وتحليل الكود تلقائياً استناداً إلى وصف من كلام الإنسان. هذه العملية تجسد بشكل أساسى نوّط من التفسير الدلالي. الدافع وراء إنشاء الكود البرمجي يمكن في إمكاناته لمساعدة مهندسي البرمجيات والمطورين في تحقيق تفاصيل عملية مع تقليل من منحني التعلم المرتبط بلغات البرمجة والتقنيات الناشئة. غالباً ما تتطلب منهجيات تطوير البرمجيات التقليدية منهاً ما تستلزم العمل الشاق لإنشاء الكود وإصلاح الأخطاء، مما يؤدي إلى عمليات ممندة ومعرضة للأخطاء.

في مجال البرمجة المساعدة بواسطة الذكاء الصناعي، يتم تسليط الضوء على أهمية إنشاء الكود البرمجي من خلال قدرته على التعامل بفعالية مع تعقيدات البرمجة. يتيح ذلك من خلال أدوات تلقائية تحليل مجموعات الكود، وفهم أنماط البرمجة، وتقديم اقتراحات أو استكمالات للكود. دمج إنشاء الكود في البرمجة المساعدة بواسطة الذكاء الصناعي يبسط مسار التطوير ويعزز جودة الكود، مما ينبع في النهاية عنه زيادة في كفاءة البرمجة وموثوقيتها.

تقدم هذه الرسالة نماذج معتمدة على المحوّلات Transformer-Based Processing لاستخدام نماذج اللغة المدرية مسبقاً Pre-trained Models على شكل المحوّلات Transformers في بناء نماذج التعلم الآلي لإنشاء الكود البرمجي تلقائياً. يمكن لـ نماذج المحوّلات أداء مجموعة واسعة من مهام البرمجة اللغوية العصبية مثل الترجمة الآلية وتحليل المشاعر والتصنيف وما إلى ذلك. اقرؤنا في هذا البحث دراسة نماذج لغة المحوّلات المدرية مسبقاً وأهميتها في بناء نماذج فك التشفير والمشفرة لتوليد الكود.

الاستفادة من نماذج اللغة المدرية مسبقاً هي استراتيجية ذكية للتغلب على مشكلة إنشاء الكود وتطوير نموذج ترجمة آلية مع إدخال اللغة الطبيعية وإخراج الكود.

لقد طورنا ونفذنا النماذج الجديدة في إنشاء الكود البرمجي، وذلك باستخدام نماذج لغوية مدربة مسبقاً لمهمات إنشاء التسلسل للحصول على نتائج أكثر دقة. تعتمد نماذجنا المقترنة على نماذج المحوّلات المدرية مسبقاً، واستخدمنا خمسة نماذج محوّلات وهي MarianMT و DistilBERT و DistilRoBERTa و ELECTRA و LUKE في تجاربنا، جميع النماذج المقترنة عبارة عن مجموعة من 6 طبقات في المشفّر ومجموعة من 6 طبقات في بنية فك التشفير. وعلاوة على ذلك، تشمل الرسالة تحليلًا للأكواد المؤلدة، بما في ذلك اكتشاف الأخطاء الثابتة ومرحلة إعادة البناء اللاحقة.

تُظهر اعتماد نماذج اللغة المدرية مسبقاً كاستراتيجية ذكية للتغلب على تحديات إنشاء الكود البرمجي، مما يؤدي إلى إنشاء نموذج ترجمة آلية ماهر قادر على تحويل الوصف البشري إلى كود برمجي قابل للتنفيذ.

أخيراً، تستكشف هذه الرسالة تنفيذ QLoRA وPEFT بالاشتراك مع نموذج اللغة (LLM) Llama 2، حيث تقوم بفحص الأسس النظرية والتداعيات العملية على إنشاء الأكواد المتعددة الأسطر بشكل شامل. وهذا ما يُقدم نموذج Llama2CodeGen، حيث يتضمن البحث تحليلاً مفصلاً لتأثيرهما على قابلية التكيف مع النموذج، وكفاءة الاستفادة من الموارد، والأداء مقارنة بنماذج إنشاء الكود المختلفة.

تؤكد النتائج التجريبية أن الاعتماد على نماذج اللغة المدرية مسبقاً على شكل المحوّلات لإنشاء الكود البرمجي هي من الحلول المهمة، مما ينبع عنه دقة وكفاءة فائقة. النماذج المقترنة، التي تم تقييمها على مجموعات بيانات CoNaLa وDJANGO، ثبتت فعاليتها. يظهر نموذج MarianCG بشكل ملحوظ متوسط BLEU بقيمة 34.43 ودقة تطابق تام بنسبة 10.2٪ على CoNaLa، بينما يقدم نتائج جاذبة على DJANGO بتقييم BLEU بقيمة 90.41 ودقة تطابق تام بنسبة 81.83٪.

بالنسبة لنموذج RoBERTaMarian، الذي هو دمج بين Marian Decoder وDistilRoBERTa، يحقق أقصى تقييم BLEU بقيمة 35.74 ودقة تطابق تام بنسبة 13.8٪ على CoNaLa، مؤكداً دقة في إنشاء الشفرة. نموذج LUKE Marian، وهو مزيج من LUKE وMarian Decoder، ينتج نتائج واحدة على مجموعة البيانات DJANGO، حيث يحصل على تقييم BLEU بقيمة 89.34 ودقة تطابق تام بنسبة 78.50٪.

بالجوهر، تسلط هذه الرسالة الضوء على قدرة نماذج اللغة المدرية مسبقاً على شكل المحوّلات على تقديم إنشاء الكود البرمجي. إنها تقدم نماذج مبتكرة قادرة على التنقل في سيناريوهات برمجية مختلفة، مما يظهر دقة وكفاءة جيدة.



مهندس: احمد شكري محمود سليمان
تاريخ الميلاد: 1990/08/08
الجنسية: مصرى
البريد الإلكتروني: ahmed.shokry.soliman@gmail.com
رقم الهاتف: +201064666545
العنوان: طوخ - القليوبية - مصر - 13742
القسم: هندسة الحاسوبات
الدرجة: ماجستير العلوم في هندسة الحاسوبات
المشرفون: أ.د/ سمير إب ارهيم محمد شاهين
أ.م.د/ مياده منصور على هدهود

الممتحنون:

أ.د/ سمير إب ارهيم محمد شاهين (المشرف الرئيسي)
أ.م.د/ مياده منصور على هدهود (المشرف)
أ.د/ رضا عبد الوهاب الخريبي (الممتحن الداخلى)
عميد كلية الحاسوبات والذكاء الاصطناعى - جامعة القاهرة
أ.د/ محمد زكي عبد المجيد (الممتحن الخارجى)
أستاذ الحاسوبات بكلية الهندسة - جامعة الأزهر

عنوان الرسالة:

نموذج تحويلي لتوليد الكود البرمجي لمساعدة المبرمجين

الكلمات الدالة:

Code Assistance; Code Generation; Transformer-Based Processing; MarianCG; Language Models

ملخص الرسالة:

إن إنشاء الكود في البرمجة المساعدة بواسطة الذكاء الصناعي يبسّط مسار التطوير ويعزز جودة الكود، مما ينبع في النهاية عن زيادة في كفاءة البرمجة وموثوقيتها. يتبع ذلك من خلال أدوات تلقائية تحليل مجموعات الكود، وفهم أنماط البرمجة، وتقديم اقتراحات أو استكمالات للكود. تقوم هذه الرسالة بنماذج معتمدة على المحوّلات Pre-trained Models لاستخدام نماذج اللغة المدربة مسبقاً Transformers في بناء نماذج التعلم الآلي لإنشاء الكود البرمجي تلقائياً. تظهر اعتماد نماذج اللغة المدربة مسبقاً كاستراتيجية ذكية للتغلب على تحديات إنشاء الكود البرمجي، مما يؤدي إلى إنشاء نموذج ترجمة آلية قادرة على تحويل الوصف البشري إلى كود برمجي قابل للتنفيذ. وأيضاً تم تحليل النتائج لهذه التجارب لعرض رسائل الخطأ وإعادة هيكلة الأكواد المولدة.

نموذج تحويلي لتوليد الكود البرمجي لمساعدة المبرمجين

إعداد

احمد شكري محمود سليمان

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة

كمجزء من متطلبات الحصول على درجة

ماجستير العلوم

في

هندسة الحاسوبات

يعتمد من لجنة الممتحنين:

♦ أ.د/ سمير إبراهيم محمد شاهين (المشرف الرئيسي)

♦ أ.م.د/ مياده منصور على هدهود (المشرف)

♦ أ.د/ رضا عبد الوهاب الخريبي (الممتحن الداخلي)

أستاذ وعميد كلية الحاسوبات والذكاء الاصطناعي – جامعة القاهرة

♦ أ.د/ محمد زكي عبد المجيد (الممتحن الخارجي)

أستاذ الحاسوبات بكلية الهندسة – جامعة الأزهر

كلية الهندسة - جامعة القاهرة

الجيزة - جمهورية مصر العربية

2024

نموذج تحويلي لتوليد الكود البرمجي لمساعدة المبرمجين

إعداد

احمد شكري محمود سليمان

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة

جزء من متطلبات الحصول على درجة

ماجستير العلوم في

هندسة الحاسوبات

تحت إشراف

أ.د/ سمير إبراهيم محمد شاهين د/ مياده منصور على هدهود

أستاذ مساعد

أستاذ

قسم هندسة الحاسوبات

قسم هندسة الحاسوبات

كلية الهندسة - جامعة القاهرة كلية الهندسة - جامعة القاهرة

كلية الهندسة - جامعة القاهرة

الجيزة - جمهورية مصر العربية

2024



نموذج تحويلي لتوليد الكود البرمجي لمساعدة المبرمجين

إعداد

احمد شكري محمود سليمان

رسالة مقدمة إلى كلية الهندسة - جامعة القاهرة

جزء من متطلبات الحصول على درجة

ماجستير العلوم

في

هندسة الحاسوبات

كلية الهندسة - جامعة القاهرة

الجيزة - جمهورية مصر العربية

2024