International Journal of Internet, Broadcasting and Communication Vol.16 No.4 394-401 (2024)
https://doi.org/10.7236/IJIBC.2024.16.4.394

IJIBC 24-4-45

Use Cases of Program Task using Tools based on Machine Learning and Deep Learning

Chae-Rim Hong[1], Jin-Keun Hong[2]*

[1]*Graduate Student, Department of AI & Bigdata, aSSIST University, Korea*
[2]*Professor, Div. of Advanced IT, Baekseok University, Korea*
E-mail: jump071383@stud.assist.ac.kr, jkhong@bu.ac.kr

Abstract

The difference of this paper is that it analyzes the latest machine learning and deep learning tools for various tasks of program such as program search, understanding, completion, and review. In addition, the purpose of this study is to increase the understanding of various tasks of program by examining specific cases of applying various tasks of program based on tools. Recently, machine learning (ML) and deep learning (DL) technologies have contributed to automation and improvement of efficiency in various software development tasks such as program search, understanding, completion, and review. This study examines the characteristics of the latest ML and DL tools implemented for various tasks of program. Although these tools have many strengths, they still have weaknesses in generalization in various programming languages and program structures, and efficiency of computational resources. In this study, we evaluated the characteristics of these tools in a real environment.

Keywords: *Application Program, Machine learning, Deep learning, Use case, Software tool*

1. INTRODUCTION

Recently, program tasks such as completing, reviewing, searching, modifying, understanding, reviewing, and optimizing programs have been rapidly developing based on machine learning and deep learning technologies. However, these studies have limited scalability due to their optimization for specific languages or program structures, and have been lacking in understanding the semantic context of programs. In addition, tools based on machine learning and deep learning require a lot of resources for calculations, and there are cases where integration is limited in automation. In this study, considering these environments, we examine the latest tools that can be used for various tasks of programs such as searching, understanding, completing, and reviewing, and identify their strengths and weaknesses. The results of the study are to understand the characteristics of machine learning tools and deep learning tools in terms of their strengths and weaknesses, and this understanding is expected to be helpful for future program work. For example, the Codex tool has excellent performance in program understanding and generation, but consumes a lot of resources. The Sourcegraph tool has excellent search performance in large program bases, but has limited performance in complex queries. Models such as Bidirectional Encoder Representations from Transformers (CodeBERT) show strengths in tasks based on semantic similarity between program and natural language, and can be applied

Manuscript received: October. 23, 2024 / Revised: October. 28, 2024 / Accepted: November. 3, 2024
Corresponding Author: jkhong@bu.ac.kr
Tel: +82-41-550-2445
Professor, Division of Advanced IT, Baeokseok University, Korea

to environments such as automatic generation of program comments. The contribution of this study is to understand the usefulness or limitations in work environments such as actual program development, review, and completion, and to increase understanding in selecting appropriate tools.

In addition, an accurate understanding of tools allows us to judge whether the tool is suitable for a specific task and understand the scope of its application.

In Chapter 2, we will examine the scope, content, and characteristics of previous researchers' research on source program analysis. In Chapter 3, we will identify machine learning and deep learning-based tools appropriate for application environments such as program search, review, and completion, and present usage cases by comparing them by application environment. Chapter 4 concludes.

2. RELATED RESEARCH

Chen et al. studied the evaluation of large language models trained on program [1]. The researchers introduce Codex, a fine-tuned Generative Pre-trained Transformer (GPT) language model, and study Python-based program writing functions. Feng et al. studied CodeBERT, a pre-trained model for programming and natural language [2]. The researchers develop CodeBERT with a transformer-based neural network structure to learn representations that support downstream applications such as natural language program retrieval and program documentation generation. Alon et al. presented a neural model to represent program fragments as program embeddings [3]. Program vectors can be used to predict semantic properties of snippets, and program is represented as a path in an abstract syntax tree. Guo et al. reported that pre-trained models for program improve the performance of program-related tasks such as program retrieval, completion, and summarization [4]. The researchers impart meaning to important programs and present GraphCodeBERT, a transformer-based pre-trained model. Alon et al. examine programs that have functions such as program summarization, documentation, and search from source program snippets, process source program as a sequence of tokens in neural network machine translation, and are interested in seq2seq models [5]. Researchers represent a program fragment as a set of paths constituting an abstract syntax tree (AST) and consider attention in decoding to restrict path selection. Guo et al. studied tools for AI-generated video software [6]. Researchers analyze PIKA Labs and RUNWAY in terms of performance, functionality, and scope of application.

3. CASES BASED ON MACHINE AND DEEP LEARNING TOOL IN PROGRAM APPLICATION

3.1 Tool Analysis using Machine Learning and Deep Learning

Table 1 explains machine learning and deep learning tools used in the areas of program completion, representation, review, synthesis, search, and summary.

Table 1. Cases of machine and deep learning tools

Program Type	Machine learning tool	Deep Learning tool
Completion	TabNine, Codota, IntelliCode	Codex, Code T5, GPT-3/-4
Representation	TabNine, Codota, IntelliCode	CodeBERT, Code2Vec, GraphCodeBERT
Review	IntelliCode, Codota	Codex, DeepCode, GPT-3/-4
Synthesis	PROSE, FlashFill	Codex, DreamCode, DeepCode
Search	Sourcegraph	CodeBERT, GraphCodeBERT
Summary	IntelliCode, Codota	CodeBERT, GPT-4

For completion, representation, review, summary of program based on machine learning tool, there are used by TabNine, Codota, and IntelliCode. TabNine, Codota, and IntelliCode are machine learning tools used for

program completion, which can be used to automate repetitive patterns. However, these tools may have low accuracy when the program structure is complex. They may also lack understanding of the context. These tools have limited expressive power in terms of program representation. In terms of program review, these tools are useful for simple bug detection, but are limited in detecting complex logical errors.

The TabNine tool can run in cloud and local environments and is integrated with Codota to support powerful program completion. Depending on the network environment, the program completion speed may be slow in the cloud [7][8]. The Codota tool was acquired by Codota in 2020, and the functions of Codota were integrated into TabNine [9][10]. The IntelliCode tool is dependent on the Visual Studio environment and provides code review, completion, and style analysis functions [11]. On the other hand, for completion, representation, review, summary of program based on deep learning tools, there are used by Codex, Code T5, GPT-3/4, CodeBERT, Code2Vec, GraphCodeBERT, DeepCode.

Codex, Code T5, GPT-3/-4 tools have excellent understanding of context and high accuracy for complex programs. These tools are highly dependent on training data and require a lot of resources for model training.

Codex tool is powerful in converting natural language into program, but it may have privacy or security issues, so caution is required when using sensitive programs [12]. CodeT5 is strong in program conversion and summarization, and requires a lot of resources for initial setup and model training [13]. CodeBERT tool provides functions for searching and summarizing based on semantic understanding of program. It is also optimized for understanding the context of program [14]. Code2Vec tool is a model that learns vector representation of source program using machine learning. This tool analyzes the abstract syntax tree (AST) of program to understand the program structure [15]. The GraphCodeBERT tool is a specialized model that considers the graph structure of the program, but when using a pre-trained model or training a new model with a customized model, a strategy should be established considering the actual usage environment [16]. The Deep Code tool automatically detects errors or vulnerabilities in the program. It is integrated with the Synk platform and used for analysis. Privacy issues should be considered for cloud-based analysis of sensitive program [17].

Also, for synthesis, search of program based on machine learning tools, there are used by PROSE, FlashFill, and Sourcegraph. These tools are good for automated program generation of specific patterns and have high accuracy within a limited range. However, they are limited for general program generation. The PROSE tool automatically learns patterns and generates program when the user provides examples. Its weakness is that the flexibility of program generation is limited and learning is limited in complex scenarios. If pattern learning is incorrect, unintended program can be generated [18]. The FlashFill tool automatically completes the remaining program when a pattern is defined through an example. It is powerful for data organization in Excel. It is difficult to apply to complex data or program structures and has a weakness that it is suitable for simple patterns [18]. The Sourcegraph tool enables efficient program search in large program bases. It has high accuracy in program search. The Sourcegraph tool is fast in program search, but the search accuracy varies depending on the program structure or data type [20][21].

Also, for synthesis, search of program based on deep learning tool, there are used by Codex, DreamCode, DeepCode, CodeBERT, GPT-4. These tools can generate program suitable for complex requirements and can convert natural language specifications into program. In program search, CodeBERT or GraphCodeBERT tools can search based on semantic similarity of program and have high accuracy for complex queries. In the case of program summary, IntelliCode or Codota tools can improve the understanding of program through program summary and can automatically generate comments. In the case of CodeBERT and GPT-4, they have high understanding of program context, can summarize, and can be applied to complex program.

The DreamCode tool can understand complex requirements and generate suitable program, but its safety in a commercial environment is not guaranteed [22][23].

3.2 Analysis of Applied Cases based on Machine and Deep Learning Tool

As shown in Table 2, machine learning and deep learning tools are used to provide program completion. An example of program completion is shown in the program snippet used. In the program snippet used, 'numbers' is a list with variables of numbers from 5 to 9. The 'for' loop sequentially iterates over each element of the 'numbers' list. When the loop is executed, the next number in the list is assigned to the 'numbers' variable. 'def calculate_sum(x, y)' defines a function named calculate_sum and takes parameters 'x' and 'y'. The 'return a + b' function returns the sum of 'x' and 'y'. The 'return' statement returns the result of the function to the calling location. The user program is being completed using GPT-4.

Table 2. Used cases of program completion using ML/DL tool

Type	Used program	Completion program case based on GPT-4
Program	numbers = [5, 6, 7, 8, 9] for number in numbers:	print(number)
	def calculate_sum(x, y):	return x + y

Table 3 is an example of program representation using Tensor Flow. The left box in Table 3 is an example of a linear regression model. Data analysis consists of data generation, model variable setting, linear model, loss function, optimization method, and learning step definition, and the program is represented based on Tensor Flow. x_data and y_data are random data, and W and b are variables to be learned. This is an example of calculating the gradient of the loss function using tf.GradientTape and updating the variables through the optimizer.

In addition, the right box in Table 3 is an example of building a CNN model based on Tensor Flow. Data construction consists of the process of CNN model layer classification, model configuration, model compilation, data loading and preprocessing, and model learning. We looked at an example of program representation based on Tensor Flow. To help understand the program representation, an example of building a CNN model using the MNIST dataset is shown. Convolutional layers and pooling layers are stacked to extract image features, and finally, classification is performed with a Dense layer.

Table 3. Used cases of program representation using Tensor flow tool

Type	Program case based on Linear regression model	Program case based on CNN model

Program	import tensorflow as tf
	import numpy as np
	
	# data generation
	x_data = np.random.rand(120).astype(np.float32)
	y_data = x_data * 0.01 + 0.5
	# mode parameter set up
	W=tf.Variable(tf.random.uniform([1],-1.0,1.0))
	b = tf.Variable(tf.zeros([1]))
	
	# Linear model
	y = W * x_data + b
	
	# Loss function
	loss = tf.reduce_mean(tf.square(y - y_data))
	
	# Optimization
	optimizer=tf.optimizers.SGD(learning_rate=0.5)
		import tensorflow as tf
		from tensorflow.keras import layers, models
		
		# Model
		model = models.Sequential([
		layers.Conv2D(32, (3, 3), activation='relu',
		input_shape=(28, 28, 1)),
		layers.MaxPooling2D((2, 2)),
		layers.Conv2D(64, (3, 3), activation='relu'),
		layers.MaxPooling2D((2, 2)),
		layers.Conv2D(64, (3, 3), activation='relu'),
		layers.Flatten(),
		layers.Dense(64, activation='relu'),
		layers.Dense(10, activation='softmax')
		])
		
		# Model comfile
		model.compile(optimizer='adam',

```
    # Trainin stage definition
    for step in range(201):
    with tf.GradientTape() as tape:
    y_pred = W * x_data + b
    loss_value=tf.reduce_mean(tf.square(y_pred-
y_data))
    grads = tape.gradient(loss_value, [W, b])
    optimizer.apply_gradients(zip(grads, [W, b]))
    if step % 20 == 0:
      print(f"Step {step}, Loss: {loss_value.numpy()},
           W: {W.numpy()}, b: {b.numpy()}")
```

```
loss='___entropy',
metrics=['accuracy'])

# Data load and preprocessing (EX: MNIST)
# x_train (x_tr), y_train (y_tr);
# x_test (x_t), y_test (y_t)
mnist = tf.keras.datasets.mnist
(x_tr, y_tr), (x_t, y_t) = mnist.load_data()
x_tr, x_t = x_tr / 255.0, x_t / 255.0
x_tr = x_tr[..., tf.newaxis]
x_t = x_t[..., tf.newaxis]

# Model training
        model.fit(x_tr, y_tr, epochs=5,
        validation_data=(x_t, y_t))
```

Table 4 shows an example of program synthesis using the Keras tool. The Keras tool is a deep learning framework in Python [24][25]. Program synthesis is the process of combining multiple S/W programs or program fragments to create a larger program. For example, a CNN model that processes image data or an RNN model that processes text data describing the image can be synthesized to create a multi-modal model, which is a composite prediction model. The model synthesis process includes image input processing and model definition processes in the CNN model, and text input processing and model definition processes in the RNN model. In the model synthesis process, the composition is model synthesis, which is the process of creating a composite model, which is the combined model that is the output of the CNN model and the RNN model, and it consists of defining the combined final model, model compilation, model training, and creating a virtual model through model training. In the composite model processing, the outputs of the CNN and RNN are combined to create a fully connected (Dense) layer composite model that ultimately predicts the probability for 10 classes.

Table 4. Used cases of program in program synthesis based Keras tool

Type	Program synthesis using Keras

Program	from tensorflow.keras import layers, models
	CNN model: image input processing
	# CNN model definition (64*64 size of image)
	image_input = layers.Input(shape=(64, 64, 3))
	x = layers.Conv2D(32, (3, 3), activation='relu')(image_input)
	x = layers.MaxPooling2D((2, 2))(x)
	...
	x = layers.Flatten()(x)
	x = layers.Dense(128, activation='relu')(x)
	
	cnn_output = layers.Dense(64, activation='relu')(x)
	
	2. RNN model: text input processing
	from tensorflow.keras import layers, models
	
	# RNN model definition
	text_input = layers.Input(shape=(100,))
	y = layers.Embedding(input_dim=10000, output_dim=64)(text_input)
	y = layers.LSTM(128)(y)
	rnn_output = layers.Dense(64, activation='relu')(y)
	
	3. Model synthesis: output combination of CNN + RNN (complex model)

```
from tensorflow.keras import layers, models

# Model complex
combined = layers.concatenate([cnn_output, rnn_output])

# Final model using combined output
z = layers.Dense(128, activation='relu')(combined)
z = layers.Dense(10, activation='softmax')(z)

# final model definition
model = models.Model(inputs=[image_input, text_input], outputs=z)

# Model comfile
model.compile(optimizer='adam', loss='__entropy', metrics=['accuracy'])

4. Model training
# Generation of virtual data
import numpy as np

image_data = np.random.random((1100, 64, 64, 3))
text_data = np.random.randint(11000, size=(1100, 100))
labels = np.random.randint(10, size=(1100, 10))

# Model training
model.fit([image_data, text_data], labels, epochs=10, batch_size=32)
```

Table 5 shows an example of program search based on DeepLearning4J (DL4J). DL4J is an open source deep learning library that implements and executes deep learning models in Java and Scala environments [26][27].

Table 5. Used cases of program in program search based on DeepLearning4J tool

Type	Program search using DeepLearning4J

Program	# Neural network set up definition, and so on : Library import
	import org.deeplearning4j.nn.conf.MultiLayerConfiguration;
	import org.deeplearning4j.nn.conf.NeuralNetConfiguration;
	....
	import org.nd4j.linalg.lossfunctions.LossFunctions;
	# Neural network basic configuration
	public class Example {
	public static void main(String[] args) {
	MultiLayerConfiguration conf = new NeuralNetConfiguration.Builder()
	# Update parameter: iteration set up
	.iterations(1)
	# Weight initialization method set up
	.weightInit(WeightInit.XAVIER)
	# Learning rate set up
	.learningRate(0.1)
	# List builder for Neural network layer definition
	.list()
	.....
	}

```
}
```

4. CONCLUSION

Through this study, we were able to confirm that machine learning and deep learning-based program work tools are useful in tasks such as program search, understanding, completion, and review. In particular, tools such as Codex showed excellent performance in program understanding and generation, but have limitations in resource consumption and generalization issues. Tools such as Sourcegraph have excellent search performance in large-scale program bases, but performance improvement is needed in complex query processing. This study analyzed tool performance in an actual development environment in terms of strengths and weaknesses. This can help researchers select appropriate tools for various program tasks. In future studies, we plan to focus on developing tools with a hybrid approach that combines the strengths of ML and DL tools, lightweight models for real-time application, and interface research.

REFERENCES

[1] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de O. Pinto, J. Kaplan, et. al, "Evaluating Large Language Models Trained on Code," https://arxiv.org/pdf/2107.03374. pp. 1-35. Jul 2021. **DOI: https://doi.org/10.48550/arXiv.2107.03374**

[2] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, et. al, "CodeBERT: A Pre-Trained Model for Programming and Natural Languages," https://arxiv.org/abs/2002.08155, pp. 1-12, Sep 2020. **DOI: https://doi.org/10.48550/arXiv.2002.08155**

[3] U. Alon, M. Zilberstein, O. Levy, E. Yahav, "code2vec: Learning Distributed Representations of Code," https://arxiv.org/pdf/1803.09473. in Proc. ACM on Programming Languages, Vol. 3, Issue POPL, No. 40, pp. 1-29, Jan 2019. **DOI: https://doi.org/10.1145/3290353**

[4] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, et. al, "GraphCodeBERT: Pre-training code Representations with Data Flow," in Proc. ICLR2021 9th, https://arxiv.org/abs/2009.08366, pp. 1-18, Sep 2021. **DOI: https://doi.org/10.48550/arXiv.2009.08366**

[5] U. Alon, S. Brody, O. Levy, E. Yahav, "code2seq: Generating Sequences from Structure Representations of Code," https://arxiv.org/abs/1808.01400, in Proc. ICLR2019, pp. 1-22. **DOI: https://doi.org/10.48550/arXiv.1808.01400**

[6] B. Guo, X. Shan, J. Chung, "A Comparative Study on the Features and Applications of AI Tools – Focus on PIKA Labs and RUNWAY," **The Journal of The Institute of Internet, Broadcasting and Communication (JIIBC), Vol. 16, No. 1, pp. 86-91,** Feb 2024. **DOI: https://doi.org/10.7236/IJIBC.2024.16.1.86**

[7] TabNine, *https://www.tabnine.com/*.

[8] TabNine, *https://github.com/codota/TabNine*

[9] Codota, *https://www.tabnine.com/*.

[10] Codota, *https://github.com/codota*

[11] MS IntelliCode, *https://visualstudio.microsoft.com/ko/services/intellicode/*

[12] OpenAI Codex, *https://openai.com/*

[13] Salesforce Research CodeT5, *https://github.com/salesforce/CodeT5*

[14] MS Research, *https://github.com/microsoft/CodeBERT*

[15] Technion Israel Institute of Technology, *https://github.com/tech-srl/code2vec*

[16] MS Research, *https://github.com/microsoft/graphcodebert*

[17] Snyk, *https://snyk.io/platform/deepcode-ai/*

[18] MS Research, *https://github.com/microsoft/prose*
[19] MS, *https://www.scribd.com/document/632501043/Flash-fill-1*
[20] Sourcegraph, *https://sourcegraph.com/*
[21] Sourcegraph, *https://github.com/sourcegraph/sourcegraph*
[22] DreamCode, *https://github.com/DreamPoland*
[23] DreamCode, *https://www.dreamcode.io/*
[24] Keras, *https://keras.io/*
[25] Keras, *https://github.com/keras-team/keras*
[26] Deeplearning4J, *https://github.com/deeplearning4j*
[27] Deeplearning4J, *https://github.com/deeplearning4j/deeplearning4j*