International Journal of Smart Computing and Artificial Intelligence
International Institute of Applied Informatics
2022, Vol. 6, No. 1, IJSCAI645

Program Comment Generation with Improved Distributed Representation by Seq2seq Model Using Parse Tree Information

Sakuei Onishi [*], Fumihiro Yukimoto [†], Hiromitsu Shiina [‡]

Abstract

Comments in a program's source code are important for understanding the program. Understanding the logical flow and overall procedure of the programs is important as the next step especially for beginners learning programming language, and it is inferred that appropriate comments on the source code can support it. In this study, we generate comments for source code using a distributed representation of line dependencies constructed with Word2Vec and using parse tree information obtained from the source code as input. Also, we generate comments not only for each line of source code but also for blocks, which are the logical units of processing.

Keywords: Programming learning, Comment Generating, Seq2seq Model, Encoder-Decoder Translation Model, Distributed Representation, Parse Tree

1 Introduction

To have an in-depth understanding of information technology, which is the core fourth industrial revolution, programming education will be introduced in Japanese elementary schools from 2020[1][2]. Following the introduction of programming education, interest in its teaching materials and support is increasing. Programming education for beginners is also provided at various universities[3]. As a related study, there is research on programming education for beginners using operation logs [4][5].

Understanding the logical flow and overall procedure of programs is considered to be important for learning the program. Appropriate comments in the source code are particularly helpful in learning a programming language. Thus, presenting appropriate comments for each procedure and using them for learning is important. Moreover, during code review in program development or collaborative learning, an explanation of the source code is conducted, and it is considered to be effective for nurturing the ability to think logically for programming. It is important to properly explain the segmentation and integration of programming procedures in a natural language during the idea phase. In other words,

[*] Graduate School of Informatics, Okayama University of Science, Okayama, Japan
[†] Benesse InfoShell, Okayama, Japan
[‡] AFaculty of Informatics, Okayama University of Science, Okayama, Japan

support in linguistic performance is important. A learning system is being developed that enables the swapping of algorithm procedures for tablet PCs to aid in the understanding of the procedure to solve this problem. However, the generation of procedures has problems such as vibration in representation, generation of a large number of procedures, and adjustment of representation level, and until now, studies on generating comments directly from the source code have been conducted. This study generates program comments by using parse tree information. Encoder–Decoder translation model, which is one of the Seq2seq models[6], is used to learn the parse tree information and comment pairs corresponding to the source code, and comments are generated for the new source code. Also, interdependency across lines of source code is deeply involved in the generation of comments in program source code. This study attempts to generate comments that capture the interdependency by using a distributed representation built-in advance through Word2Vec[7] for the input into LSTM[8] on the Encoder side and directly inputting the parse tree of the program source code to obtain the deep relationship between source codes. We also proposed a model that incorporates LSTM for the construction of Word2Vec to consider the order and dependency of the source code in the input. Also, it generates comments not only for each line of source code but also for blocks, which are the logical units of processing. To evaluate the generated comments, we use C language programs and their comments from the first programming class at the university as learning data. The comments are evaluated using a questionnaire, automatic machine translation evaluation index, and BLEU[9].

2 Relate Works

There are many studies on programming education for beginners, including proposals for online support systems or using card-type learning to teach program processing [10]. A rubric has been proposed for the curriculum of programming education. Shinkai et al[11]. have been conducting manual algorithm learning because the description and composition of the procedure are considered to be more important for nurturing programmatic thinking than programs themselves. Moreover, as previously mentioned, a learning system based on the method of swapping process procedures on tablet PCs is being developed[12]. In the field of software engineering research, there is research on the generation of nouns for programming comment generation [13].

The Encoder–Decoder model is used to automatically generate comments from the source code to understand the processing procedure proposed in this study[14]. The automatic generation of program comments using the pair of source code and comments by LSTM and the automatic generation of program comments using the external information are two examples provided so far[15]. To improve the accuracy of generating comments from the source code, variable information was extracted from the question sentence as external information and used during the generation of comments. Moreover, existing studies on comment generation using parse tree information includes a proposal on comment generation through learning of the substructure of Java programs[16]. The difference between this proposed method and existing methods is that the existing Encoder–Decoder model learned the pair of source code token and its comment, whereas this method improved the performance of comment automatic generation by changing it to a distributed representation that takes interdependency into account using parse tree information of the source code.

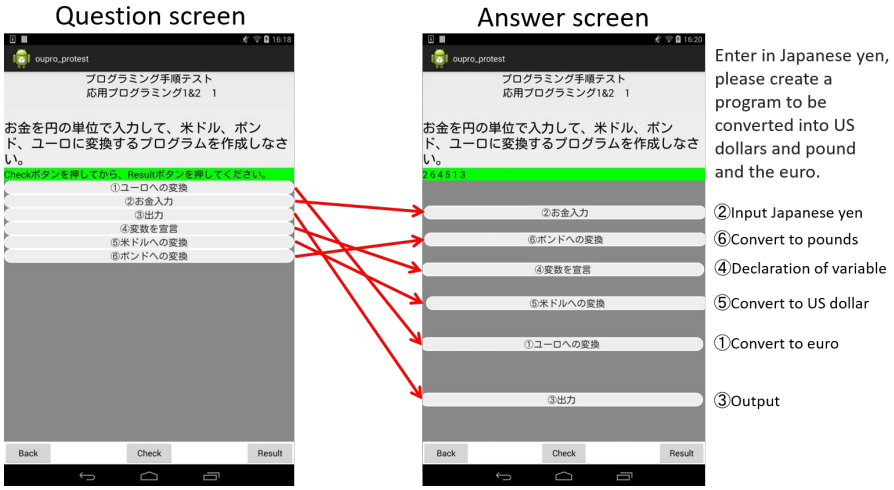

Figure 1: Procedure Learning System

```
#include <stdio.h>
int main(void){
    int yen;
    double doll;
    double pond;
    double euro;                                        ④Declaration of variable
    printf("お金を単位円で入力してください¥n");
    scanf("%d",&yen);                                   ②Input Japanese yen
    doll=yen*0.009402;                                  ⑤Convert to US dollar
    pond=yen*0.005127;                                  ⑥Convert to pounds
    euro=yen*0.007778;                                  ①Convert to euro
    printf("%d円は、%fドルです。¥n",yen,doll);
    printf("%d円は、%fポンドです。¥n",yen,pond);
    printf("%d円は、%fユーロです。¥n",yen,euro);          ③Output
    return(0);
}
```

Figure 2: Corresponding source code

3 Data Used

For this study, 30 programs in C language used in university lectures were utilized. For the line-by-line comment generation, 30 programs are used, and the total number of lines of source code is 426. Comments are added for each line of the source code. In the block-based comment generation, 51 programs are used and the total number of blocks is 352 blocks. Comments are added corresponding to each block of source code.

4 Procedure Learning System

Programming involves a number of procedures such as declaring variables, inputting values, calculation, and the output of results. We have been developing a system for programming procedures that converts source code into Japanese procedures and then rearranges the procedures to aid in understanding the concepts of the program. In the developed system, shown in Figure 1, the problem text displayed at the top of the screen and the source code with the procedures described in Japanese is displayed as an ellipse in the center of

the screen. These ellipses can be dragged and dropped, and the problem can be solved by rearranging the steps in the gray area in the center of the screen from top to bottom. After the sorting is complete, you can press the check button to check the order the questions are sorted. This test is utilized for understanding the learning situation in lectures. For example, when given a foreign currency conversion problem, and told to "create a program where you can input amounts in yen, and it will convert it to US dollars, pounds, or euros," the procedure for solving the problem can be broken down as shown in Figure 2. In this study, we have developed a system for testing the problem of rearranging this text (Figure 1). The procedure is considered to be created automatically based on the program's source code. Understanding the program requires an understanding of the source code's line-by-line processing and the blocks that integrate them into a somewhat cohesive processing unit. There are two types of block understanding: bottom-up and top-down. Because bottom-up processing is based on source code, there is a need to understand it. In this study, we aim to understand programs' processing by automatically generating procedures for line-by-line and block-by-block processing.

5 Comment Generation from Parse Tree Information by LSTM

When using the Encoder–Decoder model, input and output word sequences are used for learning. The pair of two superficial character sequences (string) is directly used, and it is considered common to learn the pair of the token sequence of the source code and comment of neural machine translation when generating comments corresponding to the program source code. However, in this study, the LSTM input on the Encoder side was changed from the token sequence of the source code to the node sequence of the parse tree, and the pair of the node sequence that corresponds to the line of the source code and the comment corresponding to that source code was learned. Moreover, while each token of the token sequence is input into the LSTM layer by converting it from a word ID to distributed representation through a 1hot vector, this proposed method inputs into the LSTM layer by separating each node of the parse tree and the pair of its external and internal information, and converting the same into a distributed representation that learned series and interdependency. The construction method of distributed representation from the parse tree is discussed in the following section.

5.1 Construction of distributed representation of parse tree information by Word2Vec

5.1.1 Acquisition of source code parse tree information through pycparser

Pycparser[17], which is the library of Python, was used to analyze the syntax of the source code in C language. The parse tree information inside and outside the node can be obtained by tracing the generated parse tree information. Figure 3 shows an example of the parse tree of a source code generated through pycparser. In the tree shown in Figure 3, the left side subtree of "While" corresponds to the conditional sentence, and the right side subtree corresponds to the processing inside the block.

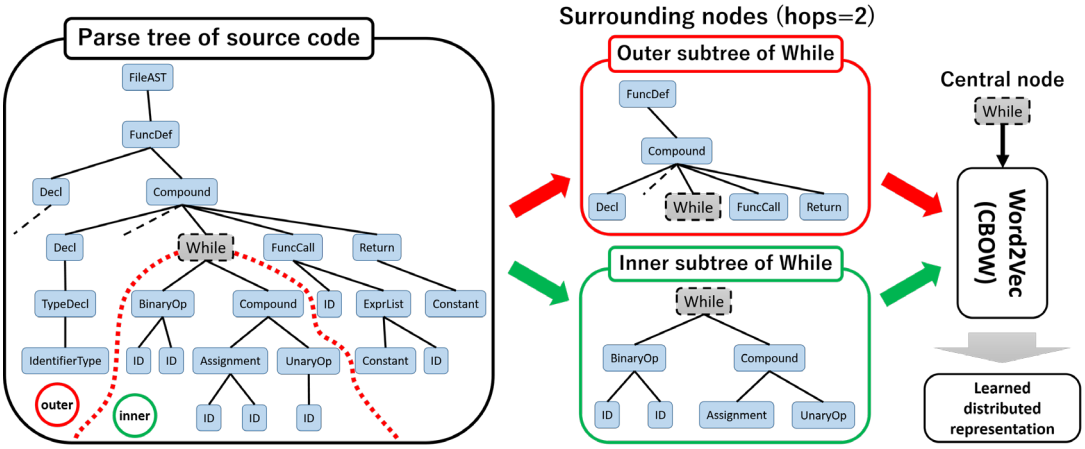

Figure 3: Construction of distributed representation of parse tree information by Word2Vec

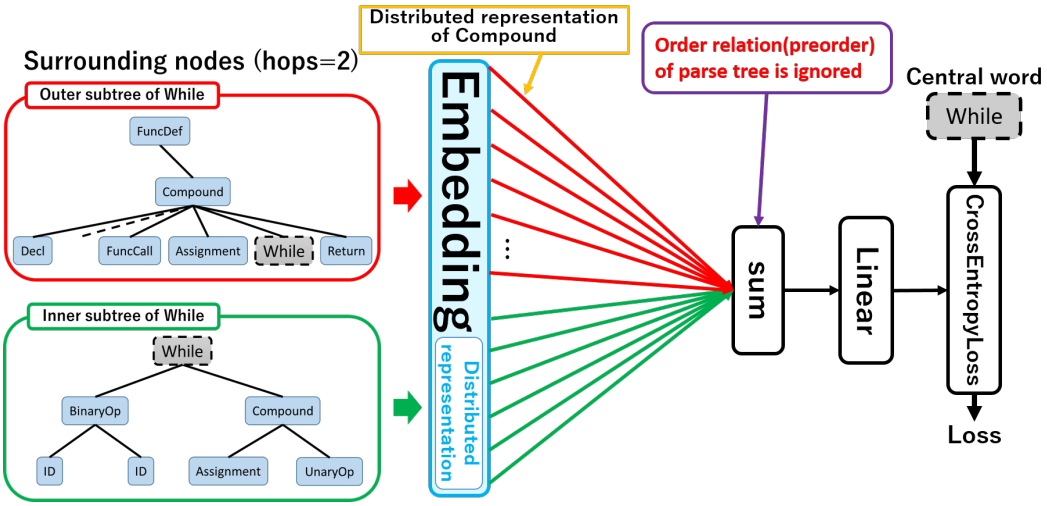

Figure 4: Word2Vec embedded

5.1.2 Construction of distributed representation

In this study, CBOW model of Word2Vec is used to construct the distributed representation of each node from the generated parse tree. The CBOW model uses a natural language corpus, such as Wikipedia, to learn the task of inferring the central word from preceding/following n-words, using each word in the corpus as the center.

However, its task was changed to infer the central node from the internal and external subtree, with each node of the parse tree as the center, and the parse tree generated from the source code as the source. The significant difference is that the preceding/following n-words in the normal CBOW model were changed to the internal and external subtree consisting of nodes whose edge number is less than hops. Note that Figure 3 shows an example of the internal and external subtree of "While" when hops=2.

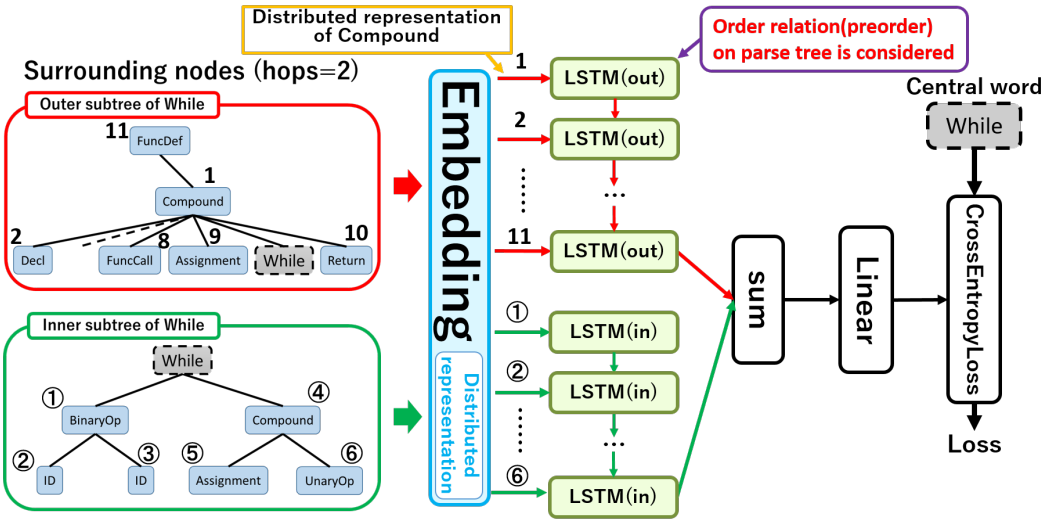

Figure 5: Word2Vec embedded with LSTM

5.1.3 Construction of distributed representation through learning of Word2Vec

Parse tree information of the inner and outer subtrees of the processing nodes described in Section 5.1.2, and the processing nodes were input to the CBOW model of Word2Vec for training. The structure of Word2Vec is shown in Figure 4. First, the nodes of the outer and inner subtrees of "While" are converted into distributed representations by the Embedding layer. Next, the distributed representations of all the nodes are summed. Since it is summing, the order relation of the parse tree is not taken into account. Finally, through the Linear layer, Loss is calculated by cross-entropy, and from Loss, back propagation is conducted to update the distributed representation. By repeating this process, the distributed representation of the nodes in the parse tree is constructed. Also, we construct a distributed representation that captures the structure of the program by considering the proximity (hops) of nodes in the parse tree.

5.2 Construction of distributed representation by learning Word2Vec using LSTM

The difference from Section 5.1.3 is that we consider the order relation in the parse tree; the structure of Word2Vec using LSTM is shown in Figure 5. First, we order the outer and inner subtrees of "While" by tracing them in the preorder traversal. Each order is indicated by a number as shown in Figure 5. Next, the nodes of the outer and inner subtrees of "While" are transformed into distributed representations by the Embedding layer. The transformed distributed representations of the nodes are input into the LSTM in the order of their numbers, and the LSTM converts them into ordered distributed representations for the outer and inner subtrees, respectively. Finally, the distributed representations of the outer and inner subtrees are summed.

5.3 Comment generation by Encoder–Decoder translation model

On the Encoder side, the Encoder–Decoder translation model converts the input into a distributed representation at the Embed layer and inputs it into the LSTM layer. At the Embed

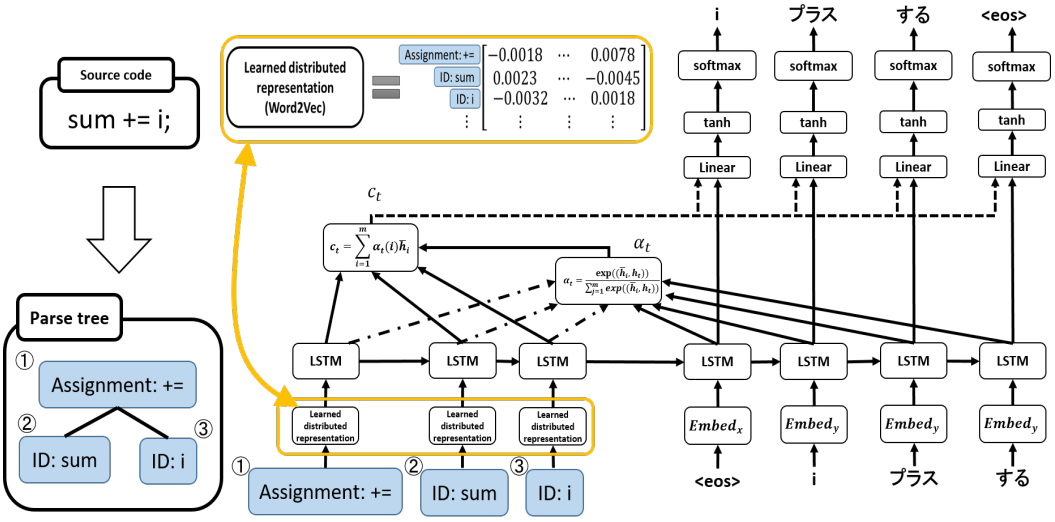

Figure 6: Encoder–Decoder translation model using learned distributed representation

layer, the Decoder side continues its learning based on distributed representation to generate correct comment input, sets the inputs as the source code or parse tree information, and uses the initial state of one–hot vector, Word2Vec, or Word2Vec embedded with LSTM for the conversion into a distributed representation. Figure 6 shows a translation model wherein the node sequence that traced the parse tree corresponding to the source code in the order of its route input to the LSTM on the Encoder side. The distributed representation constructed by Word2Vec, as discussed in Section 5.4 (Model 4), is used for the distributed representation of each node to be input into LSTM on the Encoder side. By using the trained distributed representation, which takes the nearness of nodes (hops) in the parse tree into account, comment generation that understands the program structure evaluates its possibility. Moreover, the comment generation that understands the order relation of the program evaluates the possibility by using the trained distributed representation that takes the order relation of the parse tree into account through Word2Vec embedded with the LSTM.

5.4 Comparison of distributed representations input to the LSTM on the Encoder side

To improve the comment generation of the program, the distributed representation, which is the input to the LSTM on the Encoder side of the Encoder–Decoder translation model, is changed for each model. There are two types of input: source code and parse tree. Also, there are three methods of constructing the distributed representation: initializing it with random numbers, learning the distributed representation in advance using Word2Vec, and learning it with Word2Vec incorporating LSTM. The differences in the input distributed representation for the models in the previous study and the three models proposed in this study will be discussed.

(Model1) Distributed representation initialized with random numbers of source code tokens (Figure 7(a)): To construct a distributed representation of a token in a sequence of tokens shared with the source code, the distributed representation is initialized with a random number for each token in the sequence.

(Model2) Distributed representation of nodes of parse tree information initialized with ran-

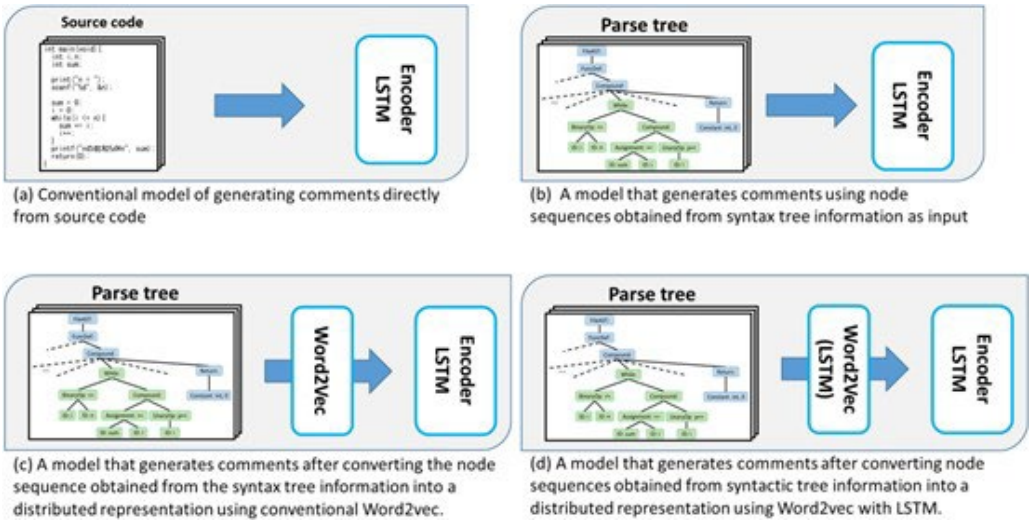

Figure 7: Comparison of distributed representation input to LSTM on Encoder side

dom numbers (Figure 7(b)): Distributed representation of nodes is constructed by initializing the distributed representation with random numbers for each node in the sequence of nodes of parse tree information corresponding to the source code.
(Model 3) Learned distributed representation constructed from parse tree information using Word2Vec (Figure 7(c)): By considering the proximity of nodes in the parse tree as described in Section 5.1.3, we can use a distributed representation that captures the structure of the program. This distributed representation is constructed by learning Word2Vec with the CBOW model (Figure 4), where each node in the parse tree is a processing node, and the task is to infer the processing node from its outer and inner subtrees. Model2 uses the initially distributed representation before learning with Word2Vec, while Model3 uses the learned Word2Vec.
(Model4) Learned distributed representation constructed with Word2Vec incorporating LSTM from parse tree information (Figure 7(d)): As in Model 3, a distributed representation is constructed by learning Word2Vec (Figure 5) with the CBOW model incorporating LSTM for the task of inferring the processing node from its outer and inner subtrees, with each node of the parse tree acting as a processing node. However, since the existing CBOW model Word2Vec (Figure 4) does not consider the order relation of inputs, we use a dis- tributed representation that considers the order relation in Word2Vec (Figure 5), which incorporates LSTM in the inference process.

6 Evaluation of Generated Comments for Each Line Using Parse Tree Information

6.1 Experimental environment for each line

As learning data, we used 30 programs written in C language used in the lectures of the University's Department of Computer Science. The model was trained in 25 programs, and the remaining 5 programs were tested. We evaluate the four models mentioned in Section 5.4. An open test is used to evaluate the generated comments. For the five different

Table 1: Overall evaluation of generated line comment

Program	Model 1	Model 2	Model 3	Model 4
P1	2.2121	3.2273	4.3939	4.1212
	(0.1185)	(0.4809)	(0.6083)	(0.6154)
P2	4.2632	5.2018	4.2281	5.0175
	(0.5115)	(0.7598)	(0.5976)	(0.7741)
P3	3.7361	2.4306	4.0833	3.8333
	(0.2427)	(0.1234)	(0.3813)	(0.3984)
P4	3.3214	4.5952	4.7738	5.4286
	(0.3440)	(0.6093)	(0.7749)	(0.8658)
P5	4.2917	3.9688	3.7083	4.4063
	(0.4952)	(0.4681)	(0.5563)	(0.5455)
Average	3.5649	3.8847	4.2375	4.5614
(BLEU)	(0.3424)	(0.4883)	(0.5837)	(0.6398)

programs, the evaluation method was based on a six-point scale from 1 to 6 by six fourth-year university students. Also, each comment of the program was evaluated using BLEU, which is an automatic machine translation index. The BLEU evaluation is based on the degree of agreement between the generated comments and correct comments and is rated in the range of 0 to 1. The closer they are to one, the higher the degree of agreement between the comments. We evaluate the comments for each program separately because the BLEU evaluation is not suitable for evaluating short sentences such as line–by–line comments.

6.2 Evaluation of generated comments for each line

6.2.1 Outline of Generated comments evaluation for each line

The questionnaire and BLEU evaluation for each program are shown in Table 1. The questionnaire evaluations are the average of the line-by-line evaluations for each program, whereas the BLEU evaluations in parentheses are the evaluations for each program. First, comparing Model1 with Models 2, 3, and 4, Models 2, 3, and 4, have higher evaluations because they use the node sequence of the parse tree as the input of LSTM on the Encoder side. There is a possibility that the way the parse tree is traced has affected the evaluation, and the order in which the parse tree is traversed, rather than the order from the beginning of the source code, may be effective in generating comments. Next, comparing Model2 and Model3, Model3 outperformed in the questionnaire evaluation and BLEU evaluations, indicating that dependencies beyond the source code line are important in comment generation. Finally, Model3 and Model4 use learned distributed representations of nodes, but Model4, which inputs a distributed representation that captures the ordering relationships of nodes in Word2Vec using LSTM, is rated higher. It was confirmed that considering the order relation of the nodes when constructing the distributed representation is effective.

6.2.2 Evaluation of each line

Figure 8 shows the generated examples, and evaluations of the programs of P4 in Table 1. Lines 3, 4, 5, and 12 show Model2, 3, and 4, which use distributed representations of parse trees as input to Encoder's LSTM, have higher evaluations than Model1 in the source code.

Furthermore, Model2, which uses random numbers as the initial values of the distributed representation of the parse tree, generates understandable comments in lines 7 and 11. In the average evaluation, Model 3, which uses a distributed representation of a learned parse tree, generates an irrelevant comment, a declaration of an integer variable, which is only 0.17 better than Model 2. However, Model 4, which uses a distributed representation of a parse tree constructed with Word2Vec with LSTM, generates more accurate comments, with an average evaluation of 0.83 better than Model 2. This improvement is thought to be the result of taking into account the order of the parse tree and the dependency between nodes.

	Source code	Model 1	Evaluation	Model 2	Evaluation	Model 3	Evaluation	Model 4	Evaluation
1	int main (void){	Declaration of main function main関数の宣言	6.0000	Declaration of main function main関数の宣言	6.0000	Declaration of main function main関数の宣言	6.0000	Declaration of main function main関数の宣言	6.0000
2	int i, n;	Declare integer type variables i and n 整数型の変数iとnを宣言	5.8333	Declare integer type variables i and n 整数型の変数iとnを宣言	5.8333	Declare integer type variables i and n 整数型の変数iとnを宣言	5.8333	Declare integer type variables i and n 整数型の変数iとnを宣言	5.8333
3	int fact;	Input process 入力の処理	1.0000	Declare integer type variable fact 整数型の変数factを宣言	5.8333	Declare integer type variable fact 整数型の変数factを宣言	5.8333	Declare integer type variable fact 整数型の変数factを宣言	5.8333
4	printf ("n=");	n	2.1667	Display n= n=を表示	6.0000	Display n= n=を表示	6.0000	Display n= n=を表示	6.0000
5	scanf ("%d", &n);	Input process 入力の処理	5.6667	Input process 入力の処理	5.6667	Input process 入力の処理	5.6667	Input process 入力の処理	5.6667
6	fact = 1;	Display 1 1を表示	1.8333	Assign 1 1を代入	5.6667	Assign 1 1を代入	5.6667	Assign 1 1を代入	5.6667
7	i = 1;	*{No output}	1.0000	Assign 1 1を代入	5.8333	整数型変数の宣言	1.1667	Assign 1 1を代入	5.8333
8	while (i <= n){	*{No output}	1.0000	If the letters are within the range of a – z 文字がa～zの範囲なら	1.0000	Repeat until it satisfies the condition 条件を満たすまで繰り返す	5.8333	Repeat until it satisfies the condition 条件を満たすまで繰り返す	5.8333
9	fact = fact * i;	Assign the product 積を代入	5.0000	Assign the product 積を代入	5.0000	Assign the product 積を代入	5.0000	Assign the product 積を代入	5.0000
10	i++;	1	1.8333	Loop until the end of the letter string 文字列の最後までループ	1.0000	Add 1 1プラスする	5.8333	Add 1 1プラスする	5.8333
11	}	Block ブロック	2.6667	End block ブロック終了	5.5000	Declare integer type variable 整数型変数の宣言	1.0000	End block ブロック終了	5.5000
12	printf ("n!=%d¥n", fact);	Input 入力	1.0000	Display n!=fact n!=factを表示	6.0000	Display n!=fact n!=factを表示	6.0000	Display n!=fact n!=factを表示	6.0000
13	return (0);	Return value 0 返り値0を返す	6.0000	Declare return value 返り値を宣言	4.0000	Return value 0 返り値0を返す	6.0000	Return value 0 返り値0を返す	6.0000
14	}	End block ブロック終了	5.5000	When it satisfies variable 変数を満たす場合	1.0000	*{No output}	1.0000	*{No output}	1.0000

Figure 8: Examples of generated comments and evaluation for each line in program (P4)

7 Comment Generation for Each Block by LSTM Using Parse Tree Information and Its Evaluation

7.1 Overview of per-block comment generation

The process of comment generation in block units is the same as that of comment generation in line units, except that the unit of input/output is changed from line units to block units. The illustration of comment generation using Encoder-Decoder translation model is shown in Figure 6, and the overview of the differences in the process of generating comments per block and per line is shown in Figure 9. The input source code

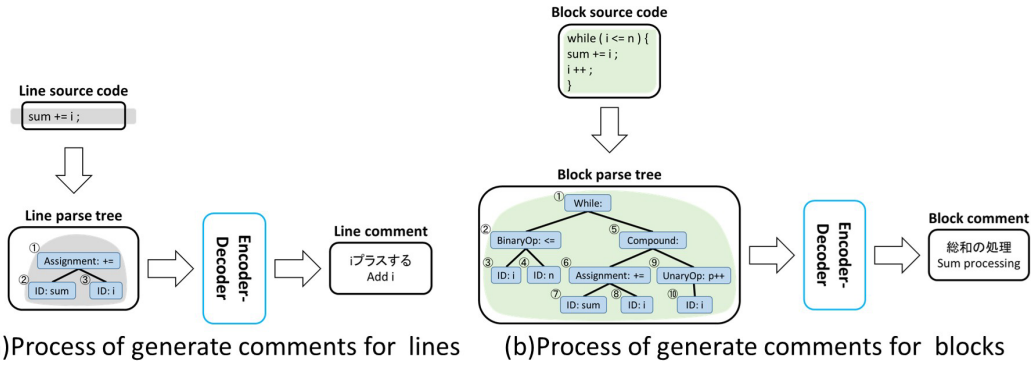

Figure 9: Differences in the process of comment generation for each block and each line

for the line–by–line comment generation in Figure 9(a) is one line, but in the case of block-by-block, four lines are input as one block in the example of Figure 9(b). The parse tree for each block corresponding to the source code input for each block is input into the Encoder–Decoder translation model. A block-wise comment such as “processing of sums” corresponding to the block is generated. The only difference between the two companies can be found in the scope of the parse treeinformation.

7.2 Experimental environment for each block

We used 51 programs in C language used in lectures of the university’s Department of Information Science as training data. We trained the model on 45 of the programs and tested it on the remaining 6. Four different models, as well as line-by-line comment generation, were evaluated. The generated comments were evaluated by open testing. For the evaluation method, six different programs were evaluated on a six-point scale from 1 to 6 by six fourth-year university students. Furthermore, the comments of each program were evaluated using BLEU, an automatic machine translation index.

7.3 Outline of comment evaluation

Table 2 shows the questionnaire and BLEU ratings for each program in terms of block-by-block comment generation. Model 1 had the lowest rating in both the questionnaire and BLEU evaluations, while Model 2 had the highest rating in both. Model 4 had a slightly lower rating than Model 2 in the line-by-line comment generation. Model 3 was rated higher than Model 1, which used source code as input, but lower than Model 2, which did not use the learned distributed representation. The input is the same as the parse tree when using the learned distributed representation, but the evaluation is a little lower. The reason for using a distributed representation of the parse tree is to capture dependencies across lines of source code. In block-by-block comment generation, the input parse tree is block-by-block, and since the input is in rough units of processing, we were able to sufficiently capture dependencies beyond lines from the block-by-block input compared to line-by-line comment generation. Also, the improvement in the evaluation of Model 4 compared to Model 3 is due to the consideration of the ordering relationship of nodes.

Table 2: Overall evaluation of generated block comment

Program	Model 1	Model 2	Model 3	Model 4
P1	5.3056 (0.4104)	5.2500 (0.4475)	4.9722 (0.4028)	4.9722 (0.3150)
P2	4.8810 (0.4074)	4.8095 (0.4632)	4.8571 (0.4632)	5.1190 (0.4632)
P3	5.0952 (0.3826)	5.1905 (0.3821)	5.2143 (0.3826)	5.3095 (0.3876)
P4	5.1667 (0.4110)	5.6111 (0.8456)	5.4444 (0.6923)	5.5556 (0.8456)
P5	4.9762 (0.5811)	5.3571 (0.5900)	4.9524 (0.5499)	5.0952 (0.6469)
P6	4.3750 (0.3917)	4.9583 (0.3938)	4.7083 (0.3790)	4.8333 (0.3765)
Average (BLEU)	4.9666 (0.4307)	5.1961 (0.5204)	5.0248 (0.4783)	5.1475 (0.5058)

7.4 Evaluation of each block

Figure 10 shows the generated examples and evaluations for the program (P4) in Table 2. In the third block, Models 2, 3, and 4, which use a parse tree as input, can correctly generate the type of the variable, “Declaration of a floating-point variable.” In the fifth block, Model 1 generates an inaccurate comment, “Output of condition,” and Model 3 fails to generate the process of being output, “Conditional judgment.” However, Model2 and Model4 were able to generate a process called “condition judgment and output,” which judges the conditions and outputs according to the conditions. The questionnaire evaluation results showed that Models 2 and 4 were highly rated. For the other lines, there is no difference in the generated block comments, and there is no significant difference in the questionnaire evaluation. Model 1 has the lowest average rating because it has the lowest ratings in two blocks, 3 and 5. Model 2 and Model 4 have the same average rating of 0.8456 by BLEU because the generated block comments are the same, but Model 2 has a slightly higher rating of 5.61111 in the questionnaire evaluation. As with the evaluations for each program, there was a significant difference between the evaluations based on the source code and the type of input to the parse tree, confirming the effectiveness of the parse tree.

8 Summary and Future Tasks

In this study, instead of pairs of source code and comments, pairs of parse trees and comments were trained as bilingual data, and program procedures were generated to understand the necessary processes and procedures for programs. In addition, by using Word2Vec to construct a distributed representation of parse tree information and using the distributed representation as input to the LSTM on the Encoder side, we attempted to generate comments that capture the structure of the program. It was confirmed that the evaluation was higher when the parse tree information was generated and input to the LSTM than when the source code was input directly to the encoder. The evaluation of the model using distributed representation that captured dependencies across lines of source code did not improve with

	Source code	Model 1	Evaluation	Model 2	Evaluation	Model 3	Evaluation	Model 4	Evaluation
1	#include <stdio.h>	Header processing ヘッダー処理	5.5000	Header processing ヘッダー処理	5.6667	Header processing ヘッダー処理	5.8333	Header processing ヘッダー処理	5.6667
2	int main(void){	Declaration of main function main関数の宣言	5.3333	Declaration of main function main関数の宣言	5.6667	Declaration of main function main関数の宣言	5.6667	Declaration of main function main関数の宣言	5.5000
3	double dx; double dy;	Declaration of input variables 入力変数の宣言	4.6667	Floating point variable declaration 浮動小数点型変数の宣言	5.5000	Floating point variable declaration 浮動小数点型変数の宣言	5.5000	Floating point variable declaration 浮動小数点型変数の宣言	5.6667

4	print f("1つ目の入力:");
	scanf("%lf",&dx);
	print f("2つ目の入力:");
	scanf("%lf",&dy);
		Input processing
		入力処理
			5.3333	Input processing
				入力処理
					5.6667	Input processing
						入力処理
							5.5000	Input processing
								入力処理
									5.6667

5	if((dx>=0&&dx<10)&&(dy>=0&&dy<10)||(dx>=20&&dx<30)&&(dy>=0&&dy<25)){
	print f("x=%f,y=%fは青色¥n",dx,dy);
	}else{
	if((dx>=10&&dx<10)&&(dy>=10&&dy<25)
	||(dx>=10&&dx<20)&&(dy>=0&&dy<10)){
	print f("x=%f,y=%fは黄色¥n",dx,dy);
	}else{
	print f("x=%f,y=%fは白色¥n",dx,dy);
	}
	}
		Condition output
		条件の出力
			4.6667	Condition judgment and output
				条件判定と出力
					5.8333	Condition judgment
						条件判定
							5.0000	Condition judgment and output
								条件判定と出力
									5.5000

6	return(0); }	End of function 関数の終了	5.5000	End of function 関数の終了	5.3333	End of function 関数の終了	5.1667	End of function 関数の終了	5.3333

Figure 10: Examples of generated comments and evaluation for each block in program (P4)

block-wise comment generation, but it did improve with line-wise comment generation. We believe this is due to the fact that block-wise comment generation can capture cross-row dependencies from block-wise input. In addition to cross-row dependencies, the model using distributed representation, which takes into account the ordering relationship of parse tree information, received the highest rating for per-row comment generation. In the future, we would like to investigate methods to improve the evaluation in block-wise comment generation. We believe that there may be a way to fully utilize parse tree information, such as away to input node types and values separately.

References

[1] Ministry of Education, Culture, Sports, Science and Technology, "Elementary school programming education guide (2nd edition)," https://www.mext.go.jp/a_menu/shotou/zyouhou/detail/1403162.htm, 2018, accessed May. 5, 2020 (in Japanse).

[2] Ministry of Education, Culture, Sports, Science and Technology, "How to programming education at elementary school level (summary of discussion)," https://www.mext.go.jp/b_menu/shingi/chousa/shotou/122/attach/1372525.htm, 2016, accessed May. 5, 2020 (in Japanse).

[3] H. Kanamori, T. Tomoto, and T. Akakura, "Development of a computer programming learning support system based on reading computer program," in *Human Interface and the Management of Information. Information and Interaction for Learning, Culture, Collaboration and Business*, S. Yamamoto, Ed. Berlin, Heidelberg: Springer BerlinHeidelberg, 2013, pp. 63–69.

[4] K. Okimoto, S. Matsumoto, S. Yamagishi, and T. Kashima, "Developing a source code reading tutorial system and analyzing its learning log data with multiple classification analysis," *Artificial Life and Robotics*, vol. 22, no. 2, pp. 227–237, apr 2017. [Online]. Available: https://doi.org/10.1007%2Fs10015-017-0357-2

[5] S. Matsumoto, K. Okimoto, T. Kashima, and S. Yamagishi, "Automatic generation of c source code for novice programming education," in *Human-Computer Interaction. Theory, Design, Development and Practice*, M. Kurosu, Ed. Cham: Springer International Publishing, 2016, pp. 65–76. [Online]. Available: https://doi.org/10.1007/978-3-319-39510-4_7

[6] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, 2014, pp. 3104–3112.

[7] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[8] K. Greff, R. K. Srivastava, J. Koutn´ık, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[9] 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*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, Jul. 2002, pp. 311–318. [Online]. Available: https://www.aclweb.org/anthology/P02-1040

[10] S. Matsumoto, Y. Hayashi, and T. Hirashima, "Development of a card operation-based programming learning system focusing on thinking between the relations of parts," *IEEJ Transactions on Electronics, Information and Systems*, vol. 138, pp. 999–1010,08 2018.

[11] J. Shinkai, Y. Hayase, and I. Miyaji, "A trial of algorithm education emphasizing manual procedures," in *Proceedings of Society for Information Technology & Teacher Education International Conference 2016*, G. Chamblee and L. Langub, Eds. Savannah, GA, United States: Association for the Advancement of Computing in Education (AACE), March 2016, pp. 113–118. [Online]. Available: https://www.learntechlib.org/p/171656

[12] K. Sakane, N. Kobayashi, H. Shiina, and F. Kitagawa, "Kanji learning and programming support system which conjoined with a lecture," in *IEICE Technical Report*, ser. ET2014–86, vol. 114, no. 513, 2015, pp. 7–12.

[13] T. Fujiki, Y. Hayase, and K. Inoue, "Generating descriptions of nouns in software from program comments," in *IEICE*, vol. 110, no. 169, 2010, pp. 65–69.

[14] A. Takahashi, H. Shiina, R. Ito, and N. Kobayashi, "Procedure generation for algorithm learning system using comment synthesis and lstm," *International Journal of Service and Knowledge Management(IJSKM)*, vol. 3, no. 2, pp. 48–61, 11 2019.

[15] S. Onishi, A. Takahashi, H. Shiina, and N. Kobayashi, "Automatic comment generation for source code using external information by neural networks for computational thinking," *International Journal of Smart Computing and Artificial Intelligence(IJSCAI)*, vol. 4, no. 2, pp. 39–61, 12 2020.

[16] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation with hybrid lexical and syntactical information," *Empirical Software Engineering*, vol. 25, no. 3, pp. 2179–2217, jun 2019. [Online]. Available: https://doi.org/10.1007%2Fs10664-019-09730-9

[17] E. Bendersky, "Github – eliben/pycparser: Complete c99 parser in pure python," https://github.com/eliben/pycparser, 2019, accessed Jul. 15, 2019.