

Hybrid Estimation for Open-Ended Questions with Early-Age Students' Block-Based Programming Answers

Hao Wu

Tianyi Chen

Xianzhe Luo

Canghong Jin

Yun Zhang

Minghui Wu

Zhejiang University City College

31701082@STU.ZUCC.EDU.CN

31701007@STU.ZUCC.EDU.CN

31701013@STU.ZUCC.EDU.CN

JINCH@ZUCC.EDU.CN

YUNZHANG@ZUCC.EDU.CN

MHWU@ZUCC.EDU.CN

Editors: Vineeth N Balasubramanian and Ivor Tsang

Abstract

Block-based programming is of great significance for cultivating children's computational thinking. However, due to the following challenges, it is difficult to evaluate students' programming ability in online learning systems: 1) compared with the traditional Online Judge (OJ) system, there is no standard answer for a given task in block-based programming; 2) in order to promote students' interests, although the programs are not totally correct and unrelated to the task, the teacher will give a comparatively higher score. Therefore, current approaches involving output comparison and code analysis do not work effectively. Furthermore, deep learning methods also suffer from the problem of how to represent block code for classification. We propose a novel hybrid estimation model to address these challenges. We first learn graph embedding from the parsed Abstract Syntax Tree (AST) to present the logicity of the code. Next, we provide some methods to measure the workload and complexity of the code. Then, we extracted some key variables and task-irrelevant properties, introduced teacher bias. Finally, XGBoost was constructed for classification. Based on real-world data collected from an online Scratch platform by early-age students, our model outperforms KimCNN, ResNet-18, and Graph2Vec+XGBoost. Moreover, we provided statistical analyses and intuitive explanations to interpret the characteristics in various groups.

Keywords: Block-based programming; open-ended question; hybrid estimation model; graph embedding; program complexity

1. Introduction

Coding is a new literacy for the twenty-first century, and as a literacy, coding enables new ways of thinking and new ways of communicating and expressing ideas, as well as new ways of civic participation (Papadakis and Kalogiannakis, 2018). Evidence shows that young children can engage in core computational thinking skills as long as they are provided with a developmentally appropriate tool to support this learning (Kalogiannakis and Papadakis, 2017).

With the popularity of a new generation of block-based programming languages like Scratch (Resnick et al., 2009), Snap (Garcia et al., 2015) and Blockly (Fraser, 2015), block-

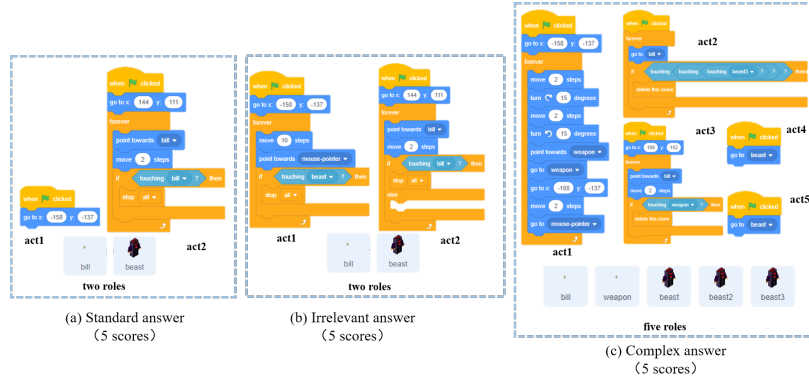


Figure 1: The answers and scores of *Bill and Beast Task*¹. The answer in (a) is to complete the procedure strictly in accordance with the requirements of this task and got 5 scores. The answer in (b) is not related to the task, but can use the knowledge node of this task proficiently, so it got 5 scores. The answer in (c), using a very complicated method to complete this task, also got 5 scores.

based programmings are becoming more commonly used to guide learners to start programming and cultivate computational thinking. These block-based programming languages build programs by snapping blocks representing statements, expression, and control structures (Maloney et al., 2010). Compared with text-based programming languages, the visual grammar of block shapes and their combination rules play the role of syntax in text-based languages: the shapes of the blocks suggest how they fit together, and the drag-and-drop system refuses to connect blocks in ways that would be meaningless, which greatly alleviates the burden of syntax for learners.

Evaluating the performance of programs is a basic task during programming education. For text-based programming education, the mature evaluating method is the OJ system. An OJ system is Web software used to compile, execute and evaluate programs submitted by users. The OJ system was originally used for programming competitions. Because the system is efficient and suitable for programming learners of different levels, they have been widely used in programming education at present (Wasik et al., 2018). For block-based programming education, the current evaluation method is to design a rule table for evaluating the level of CT and assign CT scores according to the rules (Moreno-León et al., 2015). However, block-based programming education is usually based on task-oriented learning. The limitation of the task itself leads to the limitation of the maximum CT ability that the student’s project can reflect. Therefore, CT scores can guide students in further learning, but can not directly evaluate students’ projects. For example, in the Bill and Beast¹ Task, which comes from Xiaomawang’s Scratch course series, the standard answer in Fig. 1(a), the standard answer can only get 6 CT scores, which range may range from 0 to 21 scores.

1. Bill and Beast Task: On the basis that the positions of the two roles of Bill and Beast have been initialized: First, complete the effect of Beast moving towards Bill. Second, complete the game end effect if Beast encounters Bill.

Despite the great benefits of these OJ systems in evaluating or testing human program abilities, they pose some challenges when applied to block-based programming projects as follows, take the Bill and Beast Task as an example: (i) there are no standard answers for a given task because the task is an open-ended question, as long as you complete the act of moving Beast to Bill; (ii) there may be multiple ways to accomplish this task. For example, in the realization of Beast’s function, the `control_if` code block is used in Fig. 1(a) and the `control_if_else` code block is used in Fig. 1(b); (iii) for children’s early programming education, we need a more encouraging evaluation system. Therefore, even if the child tries various ideas that exceed the task requirements (including using more roles, irrelevant functions, etc.), as shown in Fig. 1(c), the teacher still gave 5 scores.

In response to the above challenges, we propose a block-based programming estimation model for open-ended questions. In this paper, we first propose a simple yet efficient similarity indicator based on graph embedding to express the logicity of projects. We parse the projects into ASTs and learn their embedding representations at the graph level, and then calculate the similarity between the project to be evaluated and the reference projects. We used Halstead complexity measures and the cyclomatic complexity measures (Curtis et al., 1979) methods to calculate the project’s workload and complexity. In addition, we extract key variables and task-irrelevant statistics of the project as features and introduce teacher bias. Finally, XGBoost was constructed for classification. The research was based on a real-world dataset submitted effectively by more than 20,000 projects. The dataset not only contains the final project submitted by the students but also contains the scores and comments given by the teacher. The main contributions include:

- We propose a method to learn distributed representations of block-based programming projects based on ASTs, and evaluate the logicity by comparing them with the reference projects.
- We propose a hybrid estimation model for block-based programming projects containing logicity, workload, complexity, key variable, task-relevant statistics, and teachers’ bias. The features of logicity, workload, and complexity are universal, while key variables and task-irrelevant statistics depend on the requirements for the task.
- Experiments demonstrate that the proposed method has a better capability of students’ knowledge modeling. In addition, the case study shows that teachers’ subjectivity has a potential influence on rating, which facilitates teachers to improve the evaluation criteria.

2. Related Work

Evaluating how computational innovative a student is requires a standard criterion, Brennan and Resnick (2012) proposed a CT framework, which consists of three key dimensions: *computational concepts*, *computational practices* and *computational perspectives*. That allows them to assess CT’s development through three approaches, including project portfolio analysis, artifact-based interviews, and design scenarios. Wilson et al. (2012) came up with another list of criteria to evaluate each participant’s level in game coding. The coding scheme was refined based on the one that created by Denner et al. (2012), comprising of

programming concepts (like event handling and random numbers), code organization, and designing for usability (such as originality and instruction clarity). Furthermore, [Seiter and Foreman \(2013\)](#) introduced the Progression of Early Computational Thinking (PECT) Model. The model “*synthesizes measurable evidence from student work with broader, more abstract coding design patterns, which are then mapped onto computational thinking concepts.*”, aiming to disclose the correlations between age and CT ability.

To make assessments automatically, Jesús Moreno-León and Gregorio Robles et al. proposed a CT-based program evaluation method, considering the characteristic of Scratch projects as well as the concept of CT ([Moreno-León et al., 2015](#); [Boe et al., 2013](#)). Basically, this method analyses the source code of a Scratch project statically, by setting scoring criteria in various CT dimensions such as abstraction and problem decomposition, parallelism, synchronization, flow control, user interactivity, logical thinking, and data representation. Meanwhile, the detection of repetitive codes and unreasonable object naming indicates bad practices ([Moreno and Robles, 2014](#)). However, their method suffers from the inability of evaluating CT ability of recursion and debugging. [Ota et al. \(2016\)](#) designed and implemented visual programming language platform Ninja Code Village. This environment sets over sixty extensively used sample functions in advance. It runs automatic analysis to identify which functions are used to assess CT concepts like conditional statements, loops, and parallelism to foster students’ programming competencies. [Stahlbauer et al. \(2019\)](#) introduced a formal testing framework to assist programming learners and their teachers in identifying and correcting functional programming errors, since these errors may still lead to incorrect yields. The automated and property-based testing demonstrates promising capability of reproducing and replacing the manually and laboriously produced grading efforts of a teacher.

Since Scratch is still a traditional structured programming method and does not implement an object-oriented programming method, the traditionally structured program complexity measurement can analyze and evaluate Scratch programs ([Kwon and Sohn, 2016](#)). [Chang \(2019\)](#) used the McCabe method and the Halstead method to analyze Scratch projects, proposed the rules for using the McCabe metric in Scratch, used the process data in programming to verify the applicability of the Halstead method in Scratch and modify some metrics. To predict a learner’s knowledge state within a single exercise, [Jiang et al. \(2018\)](#) proposed a novel knowledge tracing method. This method computes the tree edit distance to the manually picked optimal solution to quantify a submission’s quality. [Davis \(2017\)](#) proposed a new block-based program embedding: images created from parsing the AST. The parsed ASTs are transformed into four types of images (full-color, text, and indentation; full-color, indentation, no text; text-only; blank), and then using ResNet to discuss the evaluation of block-based programming projects from the perspective of image classification.

3. Hybrid Estimation Model

In this section, we first give the formal definition of our problem. Then we introduce technical details of Hybrid Estimation Model, as shown in Fig. 2, including: 1) logical estimation model, which calculates the similarity of the graph embedding of the parsed AST to express the logicity of the project; 2) workload estimation model, which measures

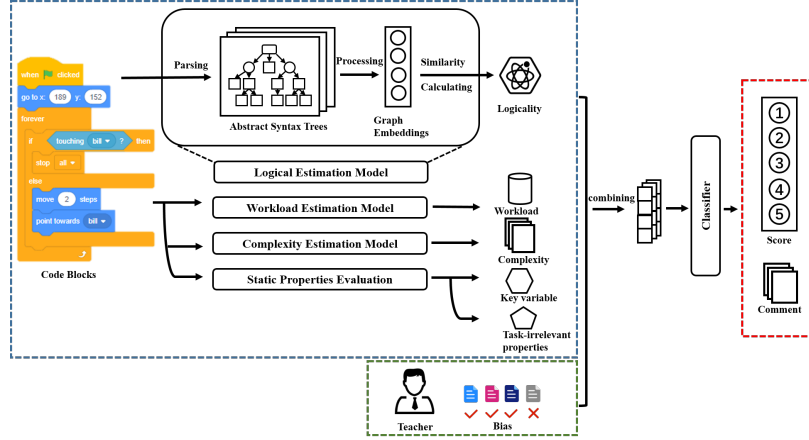


Figure 2: Overview of evaluating process, including 1) Parse and generate AST, learn graph embedding representation, and calculate similarity to express logicity; 2) Analyze the workload and complexity of the project; 3) Extract key variables and task-irrelevant properties; 4) Take the above features and the teacher’s bias as input, calculate the score of the project through the classifier and get the comment.

workload required to complete the project based on the total number of operators and operands in the program; 3) complexity estimation model, which introduces control flow as a supplement to workload estimation model; 4) static properties evaluation, which extracts key variables and task-irrelevant properties through static analysis; 5) classifier, which take the above features and the teacher’s bias as input, calculate the score the project through the classifier, in the experiment, XGBoost was used as the classifier.

3.1. Problem Definition

We introduce several basic concepts and provide a formal definition of the evaluation problem. Suppose there are n students, m teachers and p submitted projects at a learning system, which can be represented as $S = \{s_1, s_2, \dots, s_n\}$, $T = \{t_1, t_2, \dots, t_m\}$ and $P = \{p_1, p_2, \dots, p_p\}$ respectively. Each teacher will choose some projects for evaluation, and give a score of 1 to 5 and corresponding comments. Given a Scratch project completed by a student, the goal of our evaluation task is to predict the score through the Scratch project save file submitted by the student, and to explore the teacher’s evaluation criteria for Scratch visual programming.

3.2. Logical Estimation Model

3.2.1. BUILD ABSTRACT SYNTAX TREE

The AST is an abstract representation of the source code syntax structure. It expresses a programming language’s grammatical structure in a tree-like form, and each node on the tree represents a structure in the source code. In a sense, the grammar is "abstract", it

does not represent every detail that appears in the actual grammar, but only the structure or details relating to the content. Similarly, here we only focus on the Scratch Project’s logic and build an AST suitable for Scratch3.0. The specific rules are as follows: (i) take stage as the root node. In Scratch, all projects will finally show their effects in the ”stage area”; (ii) take role as the child node of the stage node, the essence of Scratch is to use code to control the role to achieve related functions. The role is the carrier of the function, which is similar to the object in object-oriented programming; (iii) the function realized by the role is regarded as the child node of the role node. The function is expressed as a series of code blocks, and the code block and the code block are connected by the logical relationship between the two.

3.2.2. GRAPH EMBEDDING

For the Graph2Vec model, the input is a complete graph corpus, and the output is a vector representation matrix of a graph with a specific dimension. First, the embeddings of all graphs in the dataset are randomly initialized, and then the rooted subgraph of each node in each graph (up to a certain degree) is extracted. Several epochs are trained to optimize the embedding of the graph.

To extract the rooted subgraph of the graph, Graph2Vec relabels it through the WL relabeling process which lays the basis for the WL kernel (Shervashidze et al., 2011). The input of the WL algorithm is the root node, the graph of the sub-graph to be extracted, and the degree of the subgraph, and the output is the rooted subgraph with the node degree d . When $d > 0$, first get all the neighbors of n (via BFS), then get the subgraph with degree $d - 1$ for each adjacent node and save it in the list, and finally get the degree $d - 1$ subgraph around the root node n is connected with the sorted list, and the expected rooted subgraph is obtained.

3.2.3. GRAPH SIMILARITY

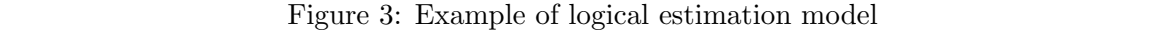
Logicity is measured by calculating each project’s similarity score and all the reference project we have given. As shown in Fig. 3, the AST parsed by the Scratch project plus the multiple reference project we have developed is used as a complete graph dataset, get the embedding of each graph through graph2vec. Then calculate the similarity according to equation (1):

$$similarity = \max \left\{ \frac{Vec_0 * Vec_i}{|Vec_0| * |Vec_i|} \right\}, \quad (1)$$

where Vec_0 is the embedding of the project, and Vec_i is the embedding of the reference project.

3.3. Workload Estimation Model

The Halstead measurement method measures the length of the program and describes the relationship between the minimum implementation of the program and the actual implementation and explains the level of the programming language based on this. It takes the operators and operands that appear in the program as the counting object. It uses their


$$\eta = \eta_1 + \eta_2, \quad N = N_1 + N_2, \quad (2)$$

where η_1 and η_2 are the number of distinct operands and the number of distinct operators, respectively. N_1 and N_2 are the total number of operands and the total number of operators, respectively. η is Program vocabulary and N is Program length.

where V is the volume, D is the difficulty and E is the Effort.

$$\mathcal{T} = \frac{E}{18} \times 1.42 + 1250, \quad O = E \times 0.007 + 46. \quad (4)$$

Since the Halstead method only considers the amount of program data and program volume, and does not consider program control flow, we introduce cyclomatic complexity as a supplement. It is a quantitative measurement of the number of linear independent paths in program source code. "Linear independence" means that each path has at least one edge that is not in one of the other paths. For example, if the source code does not contain control flow statements (conditions or decision points), the complexity will be 1, because there is only one path through the code. If there is a single-condition IF statement in the code, there will be two paths in the code: one path has the IF statement's value and the other path is FALSE, so the complexity is 2.









Scratch code block								
Operation code	operator_and	operator_or	control_if	control_if_else	control_repeat	control_forever	control_repeat_until	control_stop
Extra branches	+1	+1	+1	+2	+1	+1	+1	-1

Figure 4: The rules of cyclomatic complexity with various blocks

Mathematically, a structured program's cyclomatic complexity is defined concerning the program's control flow graph, a directed graph containing the basic blocks of the program, with an edge between two basic blocks if control may pass from the first to the second. The complexity C is then defined as:

$$C = e - n + 2b, \quad (5)$$

where e is the number of edges of the graph, n is the number of nodes of the graph and b is the number of connected components.

For a single program (or subroutine or method), b is always equal to 1. However, b is applied to several such programs or subprograms at the same time, and in these cases b will be equal to the number of programs in question, as each subprogram will appear as a disconnected subset of the graph.

The cyclomatic complexity can also be extended to programs with multiple end points. At this time, the cycle complexity is as follows:

$$C = \pi - q + 2, \quad (6)$$

where π is the number of decision points in the program and q is the number of exit points in the program.

Based on the additional branch statistics of Scratch3.0 cycle complexity, we designed an algorithm to calculate Scratch projects' cycle complexity. First, initialize the complexity to 1. Then the code block in the source code file is traversed and matched. If the corresponding code block is encountered, the addition or subtraction assignment operation is performed according to the corresponding opcode's extra branch number. The code blocks, opcodes and corresponding rules in Scratch are shown in Fig. 4. When the traversal is finished, the final loop complexity is returned.

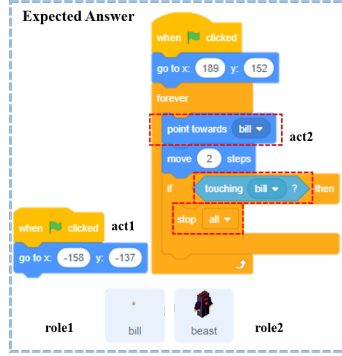
3.5. Static Properties Evaluation

We extract the key variables and task-irrelevant properties and introduce teacher as a feature shown in Table. 1, which are specific features and need to be adjusted according to different tasks.

For example, in Fig. 5, the most important things are the direction of the beast's movement, the ending action, and the condition that triggers the ending action, which corresponds to the features "towards", "exit_action" and "condition" respectively. Besides, "num_bill" and "num_beast" are classified into task-irrelevant statistics because the task requirements only include one bill and one beast. The excess part belongs to the scope of students' free play.

Table 1: Static properties table

Property Categories	Property Names	Description
Bias	teacher	Teacher who gave the evaluation
Key Variables	towards	Bill’s direction
	condition	The condition that triggers the action
	exit_action	The last action of the project
	num_ir_roles	Number of roles that are not related to the task
Task-irrelevant properties	num_ir_roles_bk	Number of blocks for roles that are not related to the task
	num_bill	Number of bill
	num_beast	Number of beast

Figure 5: The key variables of *Bill and Beast* task

4. Experiment

In this section, we first assess hybrid estimation model’s performance on the evaluation task of visual programming with several baselines. Then, we conduct a case study to visualize the explanatory power of hybrid estimation model.

4.1. Dataset Description

The experimental dataset comes from the Online Xiaomawang, an online education system developed by Hangzhou Xiaoma Education Co., Ltd. containing 20,598 real-world Scratch projects. This task requires that the positions of the two characters of Bill and Beast have been initialized: complete Beast’s effect moving towards Bill, and complete the effect of Beast if it touches Bill, the game ends. The label of the dataset is the rating outsourced to the teacher by the company, and also contains the field information of the teacher, comment, and student. As a result, all the projects are labeled with five level in the dataset, and one teacher labels each labeled (given) project.

Table. 2 shows the basic statistics of the dataset and Table. 3 represents the number distributions of roles, code blocks, code block types, and labels in the project. We can observe that: (i) more than 99% of the projects scored higher than 2 points, of which

Table 2: The statistics of the dataset

Statistic	Value
number of projects	20598
number of different labels	5
number of block types for all projects	161
average number of roles for all projects	3.31
average number of blocks for all projects	15.18
average number of block types for all projects	9.92

Table 3: The distribution of indicators in the dataset

Indicators	Categories	N	%	Indicators	Categories	N	%
Label (score)	5	15205	73.82	The number of roles	1-2	146	0.71
	4	3369	16.36		3	18390	89.28
	3	2007	9.74		4-7	1729	8.39
	2	11	0.05		≥ 8	333	1.62
	1	6	0.03				
The number of blocks	0-9	1664	8.08	The number of block types	0-8	2129	10.34
	10-11	1899	9.22		9	1206	5.85
	12	11631	56.47		10	14412	69.97
	13-16	2590	12.57		11	1125	5.46
	17-20	1146	5.56		12-15	1194	5.80
	≥ 21	1668	8.10		≥ 16	532	2.58

projects with 3, 4, and 5 points accounted for 9.7%, 16.4%, and 73.8%, respectively; (ii) about 89% of projects contain 3 roles, about 10% of projects have more than 3 roles, and less than 1% of projects have fewer than 3 roles (The reference projects contain 3 targets); (iii) about 92% of projects use no more than 20 code blocks; (iv) about 94% of projects use no more than 12 code block types.

4.2. Baseline Approaches

Our method is compared with three methods: Image Classification, Document Classification, Graph Classification. All the methods are repeated 10 times in the training and validation processes with randomly selected dataset, and the averaged results are presented.

Document Classification: Treat the source code file in the Scratch project save file as a text document, then the task becomes a traditional document classification task, we use the classic KimCNN (Kim, 2014) with pre-trained vectors from word2vec.

Image Classification: Images are created by parsing the AST. The full-color represents the code block’s specific function type, and the logical relationship reflected in the code syntax tree is indented (Davis, 2017). We tested the performance of the ResNet-18 model on the dataset.

Graph Classification: Due to the tree structure of the AST itself, we intuitively consider this task a graph-level classification. We tested the performance of Graph2Vec with XGBoost as a downstream classifier on the dataset.

Table 4: Performance evaluations with classification

Methods	Metrics				
	Precision	Recall	F1-micro	F1-macro	F1-weighted
KimCNN	0.5518	0.7386	0.7417	0.1706	0.6317
ResNet-18	0.7408	0.7046	0.7046	0.4166	0.6335
Graph2Vec+XGBoost	<u>0.7427</u>	<u>0.7765</u>	<u>0.7765</u>	0.3095	<u>0.7476</u>
Ours	0.7975	0.8124	0.8124	<u>0.3985</u>	0.7977

4.3. Parameters Setting

While computing distributed representations of graphs, we set embedding dimensions of 64; max iteration 100; learning rate 0.025; down sampling 0.0001. For XGBoost as a downstream classifier, the booster is gbtrees. The objective is multi with softmax as the loss function. We set the maximum tree depth for base learners is 6 and minimum sum of instance weight (hessian) needed in a child is 1. The max delta step is 0.32 which means the maximum delta step we allow each tree’s weight estimation. The subsample and colsample bytree are 0.85 and 0.8, respectively, representing the subsample ratio of the training instance and subsample ratio of columns when constructing each tree. The L1 regularization term and L2 regularization term on weights are 0.1 and 1e-5. The learning rate is 0.05.

4.4. Evaluation Metric

We adopt three widely used metrics, namely, precision, recall, and F1, as measures to evaluate different methods’ performance. The experimental data set is unbalanced, excluding the negligible scores 1 and 2, the sample sizes of score 3 and score 4 are only 13.20% and 22.22% compared to the sample size of score 5. Therefore, we calculated three F1 score to illustrate the effectiveness of the model, they are F1-micro, F1-macro and F1-weighted. For the F1-micro value, the number of correctly identified predictions is divided by the total number of predictions. In contrast, for the F1-macro value, it is a simple arithmetic average of the F1-score of each category. F1-weighted value is equal to the weighted average of F1-score of each category, and the weight is the proportion of each category in the true value.

4.5. Results

In this section, we present the performance of all methods on the dataset. The results are presented in Table 4 in four methods: (1) KimCNN (document classification); (2) ResNet-18 (image classification); (3) Graph2Vec+XGBoost (graph classification); (4) The proposed method. The **best** results are highlighted and the second-best results are underlined.

Based on the experimental results, we have the following observations. (1) The document classification method is weaker than other benchmark methods in most metrics. In the case of unbalanced dataset and under the premise that other metrics are weaker than other benchmark methods, although the image classification method is better than other methods on the F1-macro metric, we still believe that the performance of the image

classification method in our task is not satisfying. (2) The graph classification method is superior to other benchmark methods in most metrics. Therefore, we believe that for visual programming, treating the parsed AST as a graph and then embedding it is a new and feasible program embedding method. (3) The proposed method achieves optimal performance in precision, recall, F1-micro, and F1-weighted metrics, and F1-macro also achieves sub-optimal performance. Therefore, we believe that the method is effective.

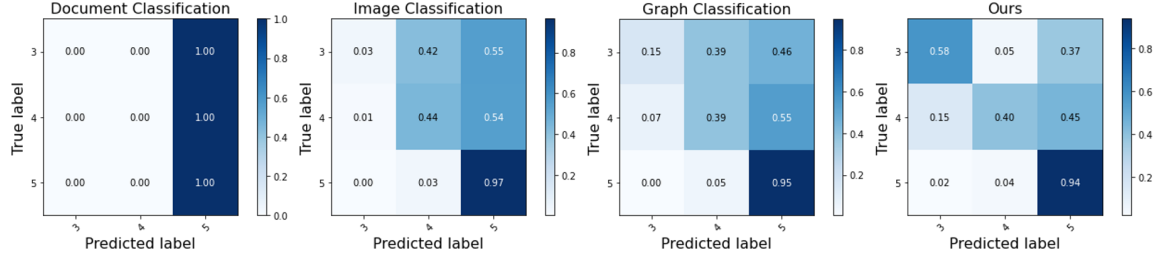


Figure 6: The confusion matrix of each method

Based on the premise of data imbalance, to further compare the performance of different methods, we draw the confusion matrix of each method (because the number of label 1 and label 2 is too small in the entire data set, they are not drawn). As shown in Fig. 6, all the predictions of the document classification method are the category of label 5 with the largest number of samples in the dataset, which does not affect. The possible reasons are complicated, especially representing the visual programming project as a document has not been finalized. Image classification and graph classification can predict the category of label 4 to a certain extent, and the correct prediction ratios are 0.44 and 0.39 respectively. However, the category of label 3 cannot be predicted well. Compared with the former two methods, the correct ratio of predicting label 4 and label 3 is increased from 0.15 and 0.39 to 0.58 and 0.40, respectively, which has a significant performance improvement.

4.6. Model Interpretation and Case Study

Previous experiments demonstrate that the proposed method has significantly outperformed baselines. However, we still do not know the evaluation criteria of visual programming. In this section, our goal is to explain our model and provide a basis for visual programming’s evaluation criteria.

4.6.1. MODEL INTERPRETATION

Before SHAP was widely used, researchers usually used feature importance to explain XG-Boost. Feature importance can intuitively reflect the importance of features and see which features have a greater impact on the final model. However, it is impossible to judge the relationship between the features and the final prediction result. The biggest advantage of SHAP value is that SHAP can reflect the influence of each sample’s characteristics, and it also shows the positive and negative effects of the impact (Lundberg and Lee, 2017). We use SHAP to explain the prediction, as shown in Fig. 7, we have the following observations. (1) Among all the characteristics, teacher, similarity, mission-related key variables, operation amount, and mccabe score are the most important. Besides, code blocks that are not

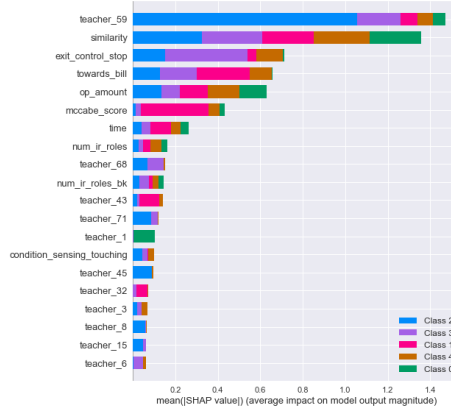


Figure 7: SHAP values of features for XGBoost in all dataset

related to the task also have a certain contribution. (2) As shown in Fig. 8, we can see that most teachers' evaluation have obvious biases. For example, in the SHAP value of label 5, the teacher "teacher_3" prefers to score 5 points, while the teacher "teacher_43" prefers not to score 5. The same situation also appears in the samples of other scores. As the challenges we mentioned in the section 1, due to the openness of the tasks and the encouragement of education, the evaluation of visual programming is affected by the teacher's subjective ideas to a considerable extent. In fact, after introducing the feature of teacher's bias, the performance has improved considerably. In addition, when applying Hybrid Estimation Model, it provides the possibility to flexibly choose different teachers according to the needs (for example, choose a teacher with a better level).

(3) Observing mission-related key variables such as "to_bill", "control_stop", and "sensing_touchingobject", we can find that the correct key variable will increase the rating, and the wrong key variable will cause the rating to drop. (4) We perceive that the more similar it is to the standard answer, the easier it is to get a high score; it is easier to get a high score with a moderate amount of operation and McCabe score. This finding is consistent with intuition: task-oriented project is naturally based on completing tasks, and the amount of operation and McCabe score should also be moderate. (5) It is worth noting that a role not related to the task and the code block on that role will also increase the rating. In this regard, we will further explore the reasons in the case analysis.

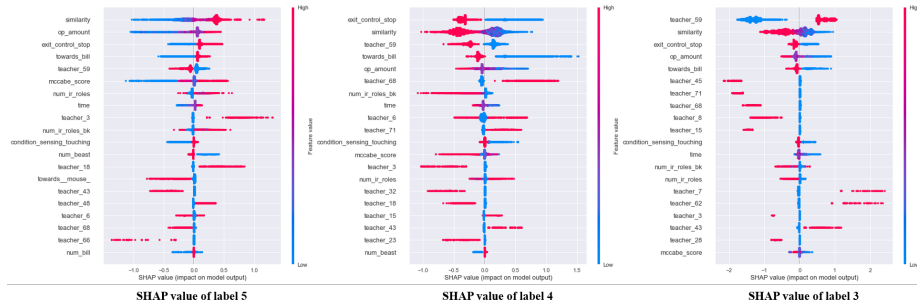


Figure 8: SHAP values of features for XGBoost on different labels

4.6.2. CASE STUDY

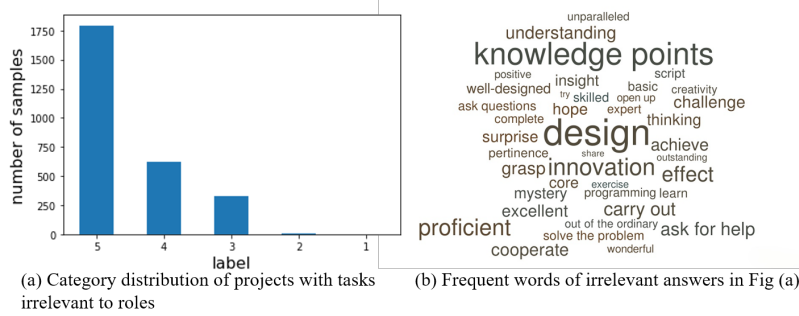


Figure 9: Incentive evaluation strategies

Here, we present the category distribution of all projects with task irrelevant roles as shown in Fig. 9(a). They have an average of 2.91 irrelevant roles and 19.1 code blocks for irrelevant roles, and the average similarity with the standard answer is 0.47. These projects with unrelated roles are not similar to the standard answers to the tasks, but they still get high points. To explore the reasons, we segmented the comments of these items and counted the word frequency. The results are shown in Fig. 9(b), we can find that design, innovation, creativity, challenge and some encouraging words appear in the word cloud. It proves that in the process of visual programming education, teachers encourage children to try innovative attitudes.

5. Conclusion and Discussion

In this paper, we investigate the problem of evaluating programs for early-age students with Scratch in open-ended questions. We have proposed a hybrid estimation model with multiple components, including logicity estimation model, workload estimation model, and complexity estimation model, to detect children’s programming ability. In addition, we introduce task-irrelevant statistics and teacher bias to explore motivation and inspiration. Our model is based on abstract syntax tree, graph embedding, program complexity, and ensemble classification. Extensive experiments have shown that the performance of the graph classification model is better than that of document classification and image classification in real-world Scratch dataset. That is to say, it is an effective way to represent block-based programming projects as graph data. Besides, our model significantly exceeds the above models. We also use visualization tools to present the weights of different features, and explain why these features are more important for the evaluation of block-based programming projects.

There are several possible future directions for our work. First, we only consider the programming ability embodied in the program from logicity, workload and complexity. Other information such as study process and knowledge tracing could be used in the future. Second, we only reflect the innovation from the roles and code blocks that are not related to the task, and there is no specific quantitative method. Third, the abstract syntax tree

still cannot make full use of the structural information of the code, especially the semantic information, such as control flow and data flow.

Acknowledgments

We thank the Hangzhou Xiaoma Education Technology Co.,Ltd for providing the real-world data. This research was partly supported by the Science Foundation of State Grid under Grant (No. B311XT19006N), Natural Science Foundation of Zhejiang Province of China under Grant (No. LY21F020003), Zhejiang Key R&D Plan Project (No. 2021C01164) and National College Student's innovation and entrepreneurship training program (No. 202013021003).

References

- Bryce Boe, Charlotte Hill, Michelle Len, Greg Dreschler, Phillip Conrad, and Diana Franklin. Hairball: Lint-inspired static analysis of scratch projects. In *Proceedings of SIGCSE*, 2013.
- Karen Brennan and Mitchel Resnick. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of AERA*, 2012.
- Chong Chang. Research and implementation of assessment system for scratch project. Master's thesis, Beijing University of Posts and Telecommunications, 2019.
- Bill Curtis, Sylvia B. Sheppard, Phil Milliman, MA Borst, and Tom Love. Measuring the psychological complexity of software maintenance tasks with the halstead and mccabe metrics. *IEEE Transactions on software engineering*, (2):96–104, 1979.
- Richard Lee Davis. Predicting unit-test scores for graphic representations of simple computer programs. 2017.
- Jill Denner, Linda Werner, and Eloy Ortiz. Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education*, 58(1):240–249, 2012.
- Neil Fraser. Ten things we've learned from blockly. In *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, pages 49–50. IEEE, 2015.
- Dan Garcia, Brian Harvey, and Tiffany Barnes. The beauty and joy of computing. *ACM Inroads*, 6(4):71–79, 2015.
- Bo Jiang, Y Ye, and Haifeng Zhang. Knowledge tracing within single programming exercise using process data. In *Proceedings of ICCE*, pages 89–94, 2018.
- Michail Kalogiannakis and Stamatios Papadakis. Pre-service kindergarten teachers acceptance of “scratchjr” as a tool for learning and teaching computational thinking and science education. In *Proceedings of the 12th Conference of the European Science Education Research Association (ESERA), Research, practice and collaboration in science education*, pages 21–25, 2017.

- Yoon Kim. Convolutional neural networks for sentence classification. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014. doi: 10.3115/v1/d14-1181. URL <http://dx.doi.org/10.3115/v1/D14-1181>.
- Kil Young Kwon and Won-Sung Sohn. A method for measuring of block-based programming code quality. *International Journal of Software Engineering and Its Applications*, 10(9): 205–216, 2016.
- Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *arXiv preprint arXiv:1705.07874*, 2017.
- John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):1–15, 2010.
- J. Moreno and G. Robles. Automatic detection of bad programming habits in scratch: A preliminary study. In *Proceedings of FIE*, 2014.
- Jesús Moreno-León, Gregorio Robles, and Marcos Román-González. Dr. scratch: Automatic analysis of scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia*, (46):1–23, 2015.
- Go Ota, Yosuke Morimoto, and Hiroshi Kato. Ninja code village for scratch: Function samples/function analyser and automatic assessment of computational thinking concepts. In *Proceedings of VL/HCC*. IEEE, 2016.
- Stamatios Papadakis and Michail Kalogiannakis. Evaluating a course for teaching advanced programming concepts with scratch to preservice kindergarten teachers: a case study in greece. In *Early Childhood Education*. IntechOpen, 2018.
- Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. Scratch: programming for all. *Communications of the ACM*, 52(11):60–67, 2009.
- Linda Seiter and Brendan Foreman. Modeling the learning progressions of computational thinking of primary grade students. In *Proceedings of ICER*, 2013.
- Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.
- Andreas Stahlbauer, Marvin Kreis, and Gordon Fraser. Testing scratch programs automatically. In *Proceedings of ESEC/FSE*, 2019.
- Szymon Wasik, Maciej Antczak, Jan Badura, Artur Laskowski, and Tomasz Sternal. A survey on online judge systems and their applications. *ACM Computing Surveys (CSUR)*, 51(1):1–34, 2018.
- Amanda Wilson, Thomas Hainey, and Thomas Connolly. Evaluation of computer games developed by primary school children to gauge understanding of programming concepts. In *Proceedings of ECGBL*, 2012.