TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 17/24 pp517–528
DOI: 10.26599/TST.2023.9010016
Volume 29, Number 2, April 2024

Towards Data-Driving Multi-View Evaluation Framework for Scratch

Xiaolin Chai, Yan Sun*, and Yan Gao

Abstract: As one of the most popular visual programming languages, Scratch has a lot of evaluation around it. Reasonable evaluation can help programmers understand their projects better. At the same time, it can also provide a reference for them to browse other projects in the online community. Most of the existing evaluations on Scratch are carried from three perspectives: Computational Thinking (CT) ability, visual presentation aesthetics, and code quality. Among them, the assessment of CT and code quality is mainly carried out from the program script, while the evaluation of visual aesthetics is analyzed from the perspective of image sequences generated by project execution. The single-view evaluation focuses on the performance of a program in a certain aspect and is one-sided. In this paper, we propose a multi-view evaluation framework to integrate various evaluations using different policies. We quantitatively analyze the assessment of different views driven by data. Combined with overall evaluations that represent human opinions, we analyze their differences and connections. Through experiments, we determine the weights of different integration policies, the proposed multi-view evaluation method can generate evaluation results similar to human opinions.

Key words: Scratch; Computational Thinking (CT); visual aesthetic; static analysis; integration policy; machine learning

1 Introduction

Scratch is a popular visual programming language. On the Scratch online community, programmers can freely share their projects, view other people's public projects, and express their opinions through likes, favorites, or adaptations. In December 2022, the Scratch online community had 676 440 870 page views with 31 590 430 unique visitors. The Scratch community currently has 120 480 903 publicly shared projects and 101 588 578 registered users. Programmers can use Scratch to create stories, games, and animations, and share them with others around the world. Appropriate evaluations can help programmers examine their projects and gain a preliminary understanding of other people's projects.

Much of the early evaluation of Scratch revolves around Computational Thinking (CT)[1, 2]. CT is a thinking ability, which refers to using concepts, methods, techniques, and logical reasoning of computing and computer science to solve problems in various fields[3]. A Dynamic Weighted Evaluation System (DWES) based on CT[4] considers the impact of different project types on CT ability for the first time and evaluates the CT embodied in Scratch projects from 8 dimensions and 5 ability levels. With the development of the field of image aesthetic evaluation[5–9], there are also studies evaluating the visual execution results of Scratch projects from an aesthetic perspective. Scratch projects are accompanied by cartoon clips playing on stage during execution. The visual aesthetic analysis of visual programming can be seen as the evaluation of the aesthetic presented

• Xiaolin Chai and Yan Sun are with Department of Computer Science, Beijing University of Posts and Telecommunication, Beijing 100876, China. E-mail: xiaolin_chai@bupt.edu.cn; sunyan@bupt.edu.cn.
• Yan Gao is with Intellectual Property Institute, Shandong Polytechnic College, Jining 272067, China. E-mail: 13863756226@163.com.
* To whom correspondence should be addressed.
Manuscript received: 2023-01-10; revised: 2023-03-16; accepted: 2023-03-17

by cartoon clips. Code quality is one of the most fundamental components of the program evaluation. The building-block programming model allows Scratch to avoid common code problems in other languages. However, after a long follow-up investigation, we find some bugs and smells in Scratch that cannot be automatically avoided. These problems can be more insidious and have serious consequences. The semantic information contained in visual code images can be used for static analysis of programs.

Although many studies have evaluated Scratch programs from CT, visual aesthetics, and code quality, few studies have focused on the connections and differences between these different perspectives. The CT ability and code problem evaluations of the project are all based on the analysis of program scripts. What are the difference and connections between them? Does the visual presentation of Scratch have a significant impact on the overall evaluation of the project? These are the questions we are interested in. Through analysis, we find that there is little correlation between single-view evaluations and overall evaluations given by humans. How to predict human true opinions on projects through evaluations from different perspectives, and provide programmers with comprehensive and multi-dimensional evaluation results as a reference are exactly what we are concerned about.

Aiming at the above problems, we propose a multi-view comprehensive evaluation framework for Scratch in this paper. The overall idea of the paper is shown in Fig. 1. First, the CT ability evaluation model, visual aesthetics evaluation model, and code problem evaluation model are used to evaluate the project from multiple perspectives. We infer overall ratings of projects from human opinions collected in the online community. Then, analyze the differences and connections between them driven by data. Finally, the evaluations from the different views are synthesized into a comprehensive assessment of the project by trying various integration policies.

In summary, our main contributions are as follows:

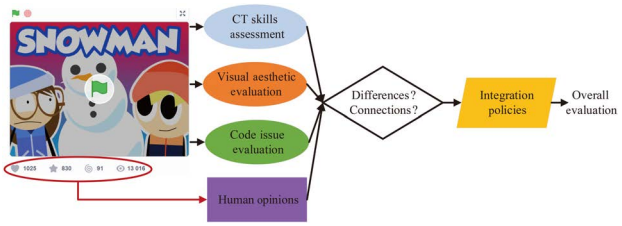


Fig. 1 Overview of our work.

- Under the data-driven, compare the differences and connections among Scratch evaluation methods from different views, and analyze the relationship between them and the overall evaluation.
- A multi-view comprehensive evaluation method is proposed, integrating evaluations from different perspectives through various policies to form a comprehensive assessment of the project.
- The weights of each evaluation of different integration policies are determined through experiments, and the connection between the generated overall evaluation and human opinions is compared.

The rest of the paper is organized as follows. In Section 2, we review Scratch evaluation methods from different perspectives. In Section 3, we analyze the relationship between various evaluations and their connections to human opinions. Then in Section 4, we introduce the multi-view evaluation method. Experimental and comparative results are given in Section 5. The conclusion of this paper is in Section 6.

2 Related Work

2.1 Computational thinking evaluation

Dr. Scratch is a web tool that helps analyze Scratch projects and assign CT scores using Hairball plug-ins[10]. It infers the CT abilities demonstrated by users from the following seven concepts: abstraction and problem decomposition, logical thinking, synchronization, parallelism, algorithmic notions of flow control, user interactivity, and data representation. In order to solve the defects of high failure rate and low efficiency in Dr. Scratch, Chang et al.[11] proposed a Scratch program analysis tool based on ANother Tool for Language Recognition (ANTLR). Their assessment of CT skills contains some basic concepts in computer science, such as stacks, queues, and recursive methods.

Considering the diversity of projects, we propose a DWES based on CT for Scratch[4]. This is a relatively comprehensive and flexible CT evaluation method. We focus on the CT ability shown by the combination of some blocks and add code organization to the evaluation dimension. Each CT concept is divided into five competency levels, as shown in Table 1. We add up the score of each dimension to get the total score of the project in CT, out of 40 points. DWES is mainly composed of two modules: project automatic analysis and score dynamic weighting, as shown in Fig. 2. The inputs of the system are the project and its type, and

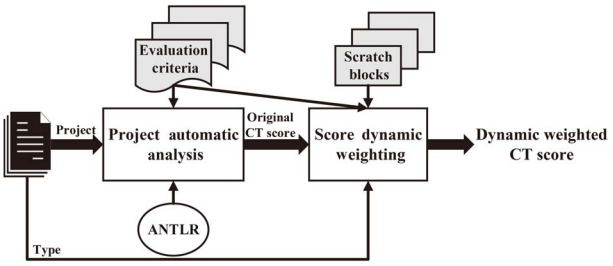


Fig. 2 Dynamic weighted evaluation system based on CT for Scratch.

the output is the dynamic weighted CT score. First, we generate the Abstract Syntax Tree (AST) of the project based on ANTLR. According to the proposed evaluation criteria, we traverse and analyze the AST to obtain the original CT score. Then, by analyzing the performance of each dimension of CT in different types of projects and the usage of blocks, we dynamically adjust the weights of each dimension of CT. This dynamic adjustment method is based on the fact that different types of projects focus on different CT skills.

2.2 Visual aesthetic evaluation

The running results of the Scratch project are presented on the stage in the form of cartoon clips. Therefore, the aesthetic evaluation of visual programming projects can be regarded as the visual aesthetic analysis of sequence cartoon images. In recent years, more and more researchers have focused on evaluating image quality from an aesthetic perspective. There are already some datasets for visual aesthetic analysis by collecting people's feedback about image quality and aesthetics on websites. The Aesthetic and Attribute DataBase (AADB)[12] and the Aesthetic Ratings from Online Data (AROD)[13] database collect images from the Flickr website. The images in AADB are scored by multiple human raters for overall aesthetics and assigned 11 attributes related to image aesthetic judgment, but the final dataset only contains 10 000 images. The AROD infers the pleasingness of an image by the number of times it has been visited and favored. The Aesthetic Visual Analysis (AVA) database contains more than 250 000 images from www.dpchallenge.com and is recognized as the benchmark dataset in the field of image aesthetic evaluation[14]. The AVA provides three types of annotations, where the aesthetic annotations of images are voted by an average of 200 amateur and professional photographers.

The emergence of large-scale datasets makes it

Table 1 CT evaluation criteria.

Competence level	Abstraction and problem decomposition	Parallelism	Logical thinking	Synchronization	Flow control	User interactivity	Data representation	Code organization
Basic (1 point)	More than one sprite and more than one script	Two scripts on green flag	If	Wait or stop all	Sequence of blocks	Say or think	Modifiers of object properties	Initialization of object properties
Developing (2 points)	Switch costumes or backdrops	Two scripts on key pressed or on the same sprite clicked	If else	A script on keyboard or mouse operation; a script when backdrop or custom switches	Repeat or forever	Green flag	Join	Rename sprites, backdrops, or costumes
Familiar (3 points)	Make a new block	Two scripts when backdrop switches	Logic operations	A script on touching color, loudness, video, timer, or object properties	Repeat until	Keyboard or mouse	Variables	Use of comment
Mastered (4 points)	Use of clones	Two scripts on loudness or video motion	Nest logical	Wait until	Loop condition contains logic operations	Webcam, input sound	Lists	No dead code
Proficient (5 points)	Use of recursion	Two scripts when receiving message	Logical nest loop	Broadcast; broadcast and wait; a script when receive message	Nest if or if else in a loop; nest loop statements in a loop	Ask and wait, answer	Queues or stacks	No useless broadcast

possible to use deep learning to evaluate image aesthetics. Lu et al.[15] proposed a double-column deep convolutional neural network to obtain global and local features of images, and combine style and semantic attributes to classify image aesthetics. Hosu et al.[16] proposed a deep learning method relying on Multi-Level Spatially Pooled (MLSP) features to evaluate the aesthetic quality of full-resolution images. Talebi and Milanfar[17] proposed a Natural Image Assessment (NIMA) method based on deep object recognition networks to predict the distribution of human opinion scores. The NIMA achieves comparable performance to other methods using a simple architecture.

To analyze and evaluate the execution results of visual programming, we proposed a model that predicts human opinion scores of sequence cartoon images[18]. Most existing research on visual aesthetics focuses on natural images, so we first construct a Cartoon Aesthetic Visual Analysis (CAVA) dataset. We propose ScratchGAN to convert photos in the AVA dataset into Scratch cartoon style and use labels reflecting relative aesthetic relations as aesthetic annotations. Based on CAVA, we propose an aesthetic evaluation method for visual programming, as shown in Fig. 3. First, the relative aesthetic prediction of a single cartoon image is obtained through the image classifier network. The image classifier network is based on Inception-Resnet-v2[19], and the last layer is replaced with a fully connected layer with 10 neurons to predict aesthetic scores in the range of 1 to 10. To verify the relative aesthetic relationship among sequence images and map them to an absolute scale, we feed the images and their relative aesthetic prediction scores into the image encoding learning network for feature extraction. On this basis, the aesthetic space is constructed to perform fine-grained sorting of images in a high-dimensional space. The l_2-norm of the sorted image features is normalized to obtain the final aesthetic evaluation of the sequence image in the range of 1 to 10.

2.3 Code problem detection

In addition to evaluating code from the perspective of CT, some research focuses on code problems in Scratch. Quality Hound[20] is an online analyzer of Scratch code

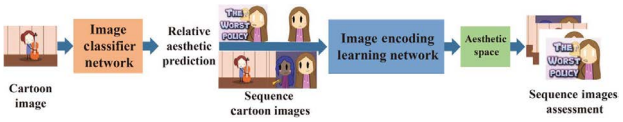


Fig. 3 Artistic analysis model based on sequence cartoon images for Scratch.

smells implemented using JastAdd. It parses the code recorded in the JSON file of the Scratch project into an AST for analysis. Quality Hound can detect 12 code smells including broad variable scope, duplicated string, duplicated code, etc. LitterBox is a linter for static analysis of Scratch programs[21]. It summarizes possible code problems in Scratch from four aspects: syntax errors, scratch-specific bugs, general bugs, and code smells. The patterns of these problems are discovered from the perspective of AST. By matching the specific patterns of these problems on the AST, the analysis results of the program are obtained.

Most methods need to convert the code of the program into an AST before analyzing, but it is time-consuming and complicated to construct the AST and traverse it once or several times to obtain the analysis results. The code images of visual programming projects contain rich semantics, which makes it possible to mine code problem patterns from the perspective of code images. We summarize the patterns of possible quality problems in Scratch from the perspective of code images, as shown in Table 2. To detect these problems, we propose a static method for visual programming languages based on code images, as shown in Fig. 4. The inputs of the method are the Scratch project and its JSON file, and the output is the analysis results of the code quality. First, code images are extracted from the project and fed into the object detection network. If no potentially problematic code patterns are detected, the static analysis is complete. If code objects are detected, we need to combine the generated AST for further analysis. Based on the code proposals, the regions in the AST that need to be analyzed are identified. Finally, we only need to analyze the important areas in the AST to get the analysis results of code problems.

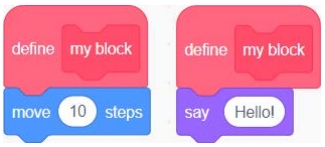
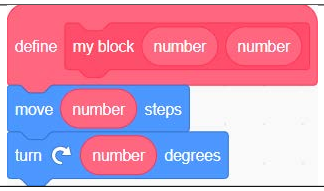
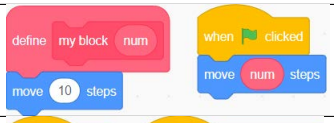
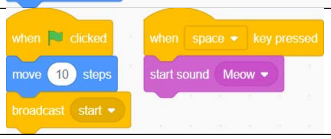
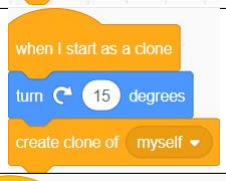
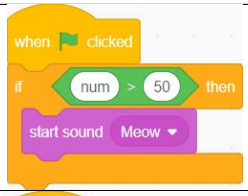
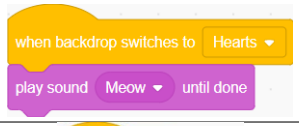
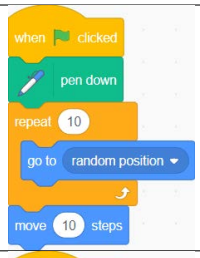
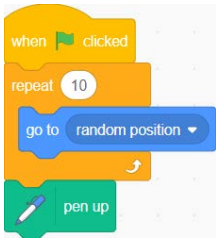
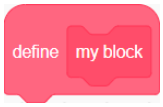
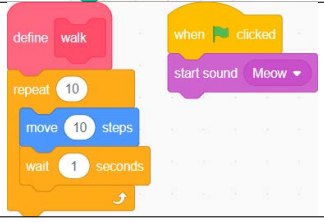
3 Towards Data-Driving Quantitative Analysis

We download more than 2500 projects from the Scratch online community during December 2022 according to their popularity. They are public and can be accessed by id. We evaluate the projects from three perspectives: CT, visual aesthetics, and code problems.

3.1 Data preprocessing

When using DWES[4] to evaluate CT, we need to dynamically adjust the weight of each dimension of CT according to the type of project, which requires the tag of the Scratch project to be correct and single. The type

Table 2 Problems and image patterns.

Problem	Code image	Problem	Code image
Ambiguous custom function name		Ambiguous parameter name	
Parameter out of scope		Unsent message	
Unreceived message		Missing clone initialization	
Missing clone call		Recursive cloning	
Illegal arithmetic operation		Missing loop	
Missing sound resources		Missing backdrop resources	
Missing costume resources		Missing pen up	
Missing pen down		Incorrect pen order	
Empty custom function		Unused custom function	
Ambiguous variable name		–	

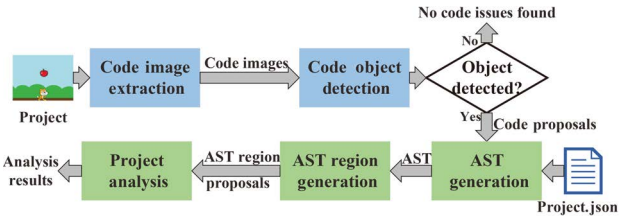


Fig. 4 Static analysis approach for visual programming.

of Scratch project is determined based on the tags selected by the user when uploading the project. We need to eliminate mislabeled projects as well as composite-type projects from the downloaded 2500 projects. After the judgment of the project classifier is combined with the selection of three domain experts, we finally choose 500 projects for analysis. The 500 projects are from animations, games, simulations, music, and stories, 100 in each category. The number of code blocks that build the program ranges from a few tens to twenty thousand.

The Scratch online community encourages users to upload their creations and share them. On the project page, visitors can run the project and express their liking for it by clicking the "like" button. Inspired by Ref. [13], we get ground-truth rating labels based on the number of views and likes of projects. The behavior of a user clicking on a project to enter the detail is counted as a view, and the behavior of a user clicking the "like" button is regarded as they love the creation. These two indicators are closely related to the overall evaluation of the project, so this is a low-threshold way to collect human feedback. We approximate the true evaluation of project p_i by

$$T(p_i) = \frac{\log L(p_i)}{\log V(p_i)} \tag{1}$$

where $L(p_i)$ is the number of likes of p_i and $V(p_i)$ is the number of views.

The evaluation we obtained using DWES is developed from 8 dimensions of CT and the total score is 40. The visual aesthetics scores obtained using the artistic analysis model range from 1 to 10. When using the static analysis method to detect code problems, the results obtained are the problems in the code and the number of times they occur. In order to conduct quantitative analysis, we need to unify the scores of these three evaluation methods in the same interval. We adjust the CT scores for a category of projects by

$$S_{\text{ct}}(p_i) = \frac{\sum_{k=1}^{8}(\text{CT}_k \times \text{PW}_{t,k}) \times \dim}{40} \times \alpha \tag{2}$$

where k indicates which dimensions of CT, t denotes the type of the project, and PW is the weight matrix. dim represents the dimension of CT involved in weighting, which is 8 here. α is used to control the score interval.

We adjust the aesthetic score interval by

$$S_{\text{aes}}(p_i) = \frac{\sum_{k=1}^{N} A_n}{10N} \times \beta \tag{3}$$

where N denotes the number of cartoon images composing the Scratch clip, A_n represents the aesthetic score of the n-th image, and β is used to control the score interval.

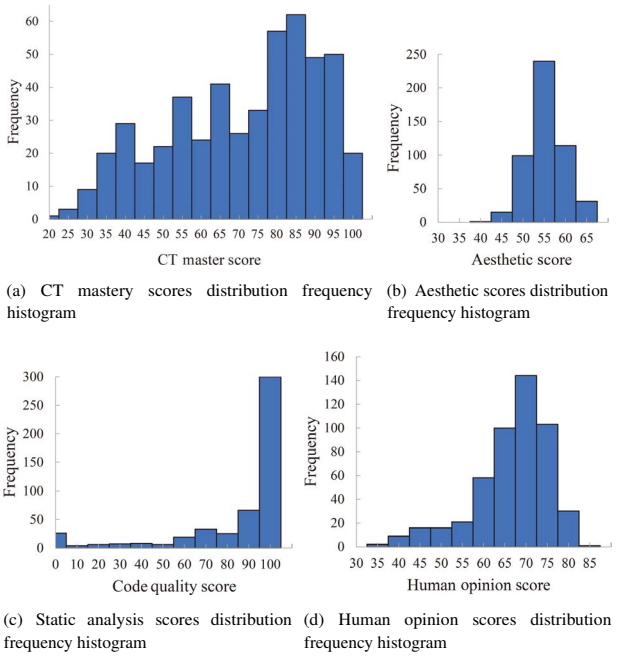
When the values of α and β are 100, we have unified the CT and aesthetic score of project p_i into the range of 0 to 100. As for the quantification of code quality detection results, we prescribe a score of 100 for projects without any problems. The static analysis score for project p_i is

$$S_{\text{pro}}(p_i) = 100 - \sum_{m=1}^{M} T_m \tag{4}$$

where M represents the number of code issues detected in p_i, and T_m represents the number of occurrences of the m-th problem.

3.2 Findings

We quantitatively analyze the CT mastery scores, aesthetic scores, code quality scores, and human opinion scores of 500 projects, as shown in Fig. 5. CT mastery scores range from 20 to 100, aesthetic scores range from



(a) CT mastery scores distribution frequency histogram (b) Aesthetic scores distribution frequency histogram

(c) Static analysis scores distribution frequency histogram (d) Human opinion scores distribution frequency histogram

Fig. 5 Frequency histograms of score distributions for the three evaluation methods and human opinions.

35.10 to 63.76, code quality scores range from 0 to 100, and human-true evaluation scores range from 30.56 to 80.14. The mean and median of CT mastery scores are 69.86 and 75, respectively. The mean and median of aesthetic scores are 53.02 and 52.81, respectively. The mean of code quality scores is 81.79 and the median is 95. The mean of human opinion scores is 64.3 and the median is 66.24. The medians of CT mastery scores, code quality scores, and human opinion scores are all greater than their means, which indicates that the distribution of scores is negatively skewed to some extent. The median and mean values of aesthetic scores are relatively close, indicating that the distribution of scores is symmetrical to a certain extent. The skewness of the distributions of the three evaluations and human opinions are all less than 0, which indicates that the distributions are skewed to the left. The absolute value of the skewness of the code quality scores is the largest, which is 1.91, indicating that the degree of left skew of the evaluation is the most serious, and the shape of the distribution is left long tail. The skewness of the human opinion scores and the CT scores are -1.20 and -0.48, respectively, which also shows that the data on the left side of the mean are more dispersed than the data on the right side. The skewness of aesthetic scores is -0.16, which is closest to 0, indicating that the data distribution is relatively symmetrical. As for data's kurtosis, the distribution of aesthetic scores is 0.46, which is relatively close to 0. The kurtosis of the CT scores is -0.83, indicating that the distribution has a gentle kurtosis and a dumpy shape. The kurtosis of the human opinion scores and the code quality scores are 1.47 and 2.68, respectively, indicating that the kurtosis of the distribution is relatively steep and pointed. Thus, it can be preliminarily judged that the distributions of CT mastery scores, code quality scores and human opinion scores do not satisfy the normal distribution, while the aesthetic scores basically conform to the normal distribution.

We further verify the normality of the distribution using the Kolmogorov-Smirnov (K-S) test[22], and the results are shown in Table 3. The p-value of the aesthetic scores is 0.63, which is greater than 0.05, satisfying the H_0 hypothesis. Therefore, the data can be considered to be normally distributed. However, the p-values of CT scores, code quality scores, and human opinion scores are 1.72×10^{-7}, 7.84×10^{-29}, and 3.43×10^{-5}, respectively, which are much smaller than 0.05 and satisfy the H_1 hypothesis, so they do not conform to normal distribution.

Table 3 Results of the K-S test.

Evaluation	p-value
CT mastery score	1.72×10^{-7}
Aesthetic score	0.63
Code quality score	7.84×10^{-29}
Human opinion score	3.43×10^{-5}

Since CT scores, code quality scores, and human opinion scores do not conform to the normal distribution, we conduct spearman correlation analysis. The correlation between various evaluation methods and real human views is shown in Table 4, and the correlation coefficients between them and real human evaluations are 0.02, 0.11, and -0.02, respectively. The absolute values of the correlation coefficients are close to 0, so we believe that there is almost no correlation between a single evaluation and the real human evaluation. This prompts us to integrate evaluations from different perspectives in a way, so that the comprehensive evaluation can be close to human opinion.

We explore the relationship between different evaluation methods and Scratch blocks, and the results are shown in Fig. 6. It can be seen that there is a strong

Table 4 Correlation between different evaluation methods and human views.

Correlation coefficient	Human opinion score
CT mastery score	0.02
Aesthetic score	0.11
Code quality score	-0.02

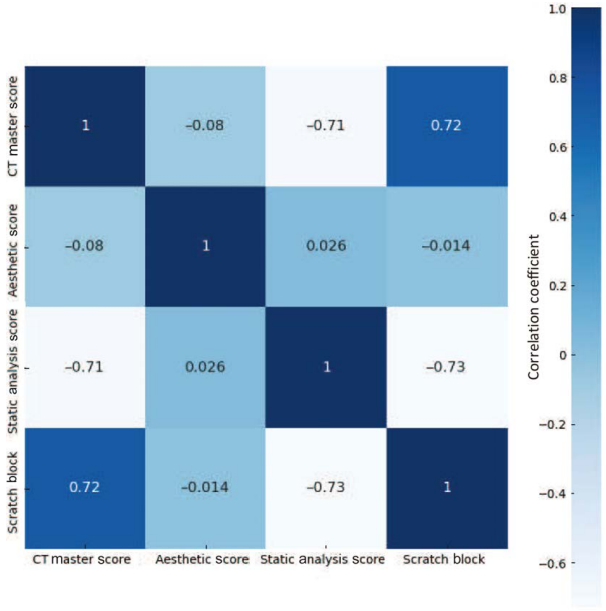


Fig. 6 Heatmap of correlation coefficients between different evaluation methods and Scratch blocks.

positive correlation between CT mastery scores and Scratch blocks with a correlation coefficient of 0.72. Quality scores for detecting code issues are strongly negatively correlated with blocks, with a correlation coefficient of −0.73. Code quality scores correlate negatively with CT scores, with a correlation coefficient of −0.71. This shows that programmers may use more CT when writing complex programs with more code blocks, but it may also cause more code problems. The artistic aesthetic score is a subjective evaluation generated by human visual perception and has nothing to do with code blocks, so the correlation coefficient is −0.01. Meanwhile, there is little relationship between aesthetic evaluation and CT or code quality evaluation.

4 Multi-View Evaluation Method

In this section, we propose a multi-view evaluation method for the visual programming language Scratch. This method obtains the overall score of the project by integrating the CT ability, artistic aesthetic, and code quality evaluation.

4.1 Framework

The proposed framework is shown in Fig. 7, where the input is the Scratch project and the output is the overall evaluation score of the project. First, the projects are extracted and preprocessed to generate scripts, sequence cartoon images, and code images required for analysis. Then, they are used as the input of various evaluation methods, and the evaluation is carried out from three views: CT ability, visual aesthetic, and code quality. Finally, the various scores are fed into the evaluation integration module for combination. We use three integration policies: simple weighted sum policy, dynamic weight policy, and machine learning-based policy. We introduce different integration policies in the following subsection.

4.2 Integration policies

The emphasis of evaluations varies from project to project, and some evaluations always play a more important role than others. For example, if a project uses a small amount of code to achieve frequent switching of backdrops and sprites, visitors will pay more attention to the visual presentation of the project and downplay its performance in terms of CT and code quality. For a project implemented with complex logic code, its visual presentation may only be a separate sprite, and the overall evaluation at this time may focus more on the functionality and quality of the code. Therefore, after using DWES, aesthetic evaluation model, and static analysis method to obtain the multi-view evaluation of the project, we propose three policies to combine them into a multi-view comprehensive evaluation of Scratch.

The first policy performs a simple static weighted sum of CT scores, aesthetic scores, and code quality scores. The overall evaluation score can be defined as

$$S_{\text{sta}}(p_i) = \delta \times S_{\text{ct}}(p_i) + \mu \times S_{\text{aes}}(p_i) + (1-\delta-\mu) \times S_{\text{pro}}(p_i) \quad (5)$$

where δ and μ are fixed weights set according to experience. δ and μ range from 0 to 1, and $\delta + \mu$ also ranges from 0 to 1.

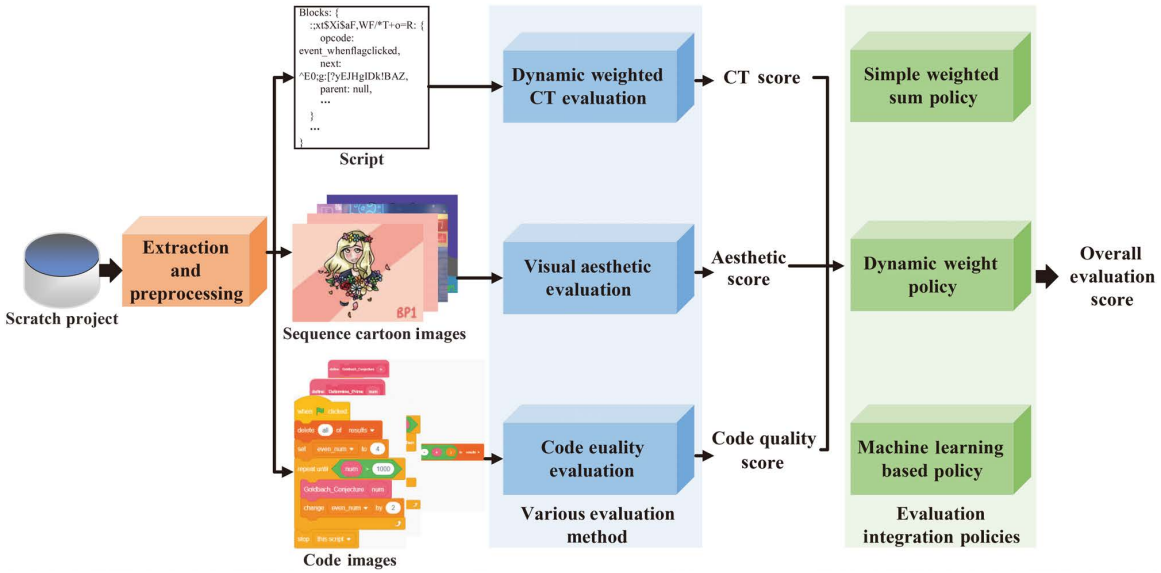


Fig. 7 Multi-view evaluation framework for Scratch.

Although the overall evaluation of projects can be obtained simply and directly through static weighted summation, there are two problems. One is that setting δ and μ may empirically is difficult. It is difficult for us to empirically set their optimal values. The other is to weight all projects with a fixed weight, ignoring the different contributions of various evaluations in different projects. Aiming at the above problems, combining with the findings in Section 3.2, we propose a second integration policy that dynamically assigns weights. Since there is a strong correlation between CT skills, and the number of code blocks in a project, projects with more code statements are likely to have higher CT mastery scores. There is a strong negative correlation between code quality and the number of code blocks, that is, projects with more code statements are likely to have more code problems. And the visual presentation of the project has little to do with the code blocks. We normalize the code number of the project as the weight,

$$W_B(p_i) = \frac{B(p_i)}{B_{\max} - B_{\min}} \tag{6}$$

where $B_{\max}$ is the number of code blocks of the project with the most codes in the dataset, and $B_{\min}$ is the number of code blocks of the project with the least code. $B(p_i)$ is the number of code blocks of project p_i. According to the use of code blocks in Scratch projects, we dynamically assign weights to the CT mastery scores, aesthetic scores, and code quality scores. The dynamic weighted overall evaluation score can be defined as

$$S_{\text{dyn}}(p_i) = W_B(p_i)\frac{S_{\text{ct}}(p_i) + S_{\text{pro}}(p_i)}{2} + (1 - W_B(p_i))S_{\text{aes}}(p_i) \tag{7}$$

The intuition behind this dynamic weighting policy is that projects with little or no code should be evaluated primarily around the visual presentation. For projects with a considerable amount of code, we should focus on observing the CT ability or existing problems reflected in the code.

Even though the above two policies are simple and intuitive, they both need to assign weights directly and are indirectly based on experience. Artificially specifying these weights requires a lot of experience to be reasonable, which is often expensive and difficult. Some evaluation methods play a more important role than others in the comprehensive evaluation of the project. Therefore, we propose a machine learning-based approach to integrate these evaluations. We use the evaluations of all methods as input features, and train the model based on the real feedback of users on the projects in the Scratch online community. Given a Scratch project, we perform dynamic weighted CT evaluation, visual aesthetic evaluation, and code quality evaluation on it to obtain scores from various perspectives. Assuming that there are a total of n projects, the evaluation of CT ability is carried out from i dimensions, the aesthetic evaluation is carried out from j dimensions, and the code quality evaluation is carried out from k dimensions. We can construct an evaluation score matrix $M = S(p_i)_{q\times(i+j+k+1)}$, as shown in Fig. 8. q represents the number of projects. i, j, and k represent the number of evaluation dimensions of CT, visual aesthetic, and code quality, respectively, TL is the ground truth label of the evaluation. Except the last column is the true evaluation of the collected projects, each row is the score of each dimension of a project, and each column is the score of one dimension for all projects. The value of each cell is obtained through different evaluation methods, that is, the evaluation score of a certain dimension.

After constructing the matrix M, the learning problem of the comprehensive evaluation is transformed into a regression problem in machine learning, that is, predicting the overall score of a project within 100 through various input evaluations. We can use any machine learning model for regression training, such as decision trees[23], random forests[24], Back-Propagation (BP) neural network[25], Gradient Boosted Decision Trees (GBDT)[26], XGBoost[27], etc. We choose XGBoost and LightGBM[28]. XGBoost is an efficient implementation of GBDT, which has strong generalization ability, high scalability, and fast computing speed. And LightGBM is an efficient implementation of XGBoost. It uses a histogram-based decision tree algorithm that discretizes continuous

$\cdots$	$\mathbf{CT_1}$	$\mathbf{CT_2}$	$\cdots$	$\mathbf{CT}_j$	$\boldsymbol{A_1}$	$\boldsymbol{A_2}$	$\cdots$	$\boldsymbol{A_r}$	$\boldsymbol{P_1}$	$\boldsymbol{P_2}$	$\cdots$	$\boldsymbol{P_t}$	$\mathbf{TL}$
$\boldsymbol{p_1}$	$S_{\text{CT}_1}(p_1)$	$S_{\text{CT}_2}(p_1)$	...	$S_{\text{CT}_j}(p_1)$	$S_{A_1}(p_1)$	$S_{A_2}(p_1)$	$\cdots$	$S_{A_r}(p_1)$	$S_{P_1}(p_1)$	$S_{P_2}(p_1)$	...	$S_{P_t}(p_1)$	$\text{TL}(p_1)$
$\boldsymbol{p_2}$	$S_{\text{CT}_1}(p_2)$	$S_{\text{CT}_2}(p_2)$	...	$S_{\text{CT}_j}(p_2)$	$S_{A_1}(p_2)$	$S_{A_2}(p_2)$	$\cdots$	$S_{A_r}(p_2)$	$S_{P_1}(p_2)$	$S_{P_2}(p_2)$	...	$S_{P_t}(p_2)$	$\text{TL}(p_2)$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
$\boldsymbol{p_q}$	$S_{\text{CT}_1}(p_q)$	$S_{\text{CT}_2}(p_q)$	...	$S_{\text{CT}_j}(p_q)$	$S_{A_1}(p_q)$	$S_{A_2}(p_q)$	$\cdots$	$S_{A_r}(p_q)$	$S_{P_1}(p_q)$	$S_{P_2}(p_q)$	$\vdots$	$S_{P_t}(p_q)$	$\text{TL}(p_q)$

Fig. 8 Evaluation score matrix *M*.

floating-point features into k discrete values and constructs a histogram with a width of k. Although we use XGBoost and LightGBM in our experiments, when more powerful regression methods are available, we only need to make a simple replacement according to the proposed framework to use the new method. We denote the evaluation score obtained by learning as S_{ml}.

5 Experiment

We use static-weighted, dynamic-weighted, and machine learning based weighted policies to integrate the evaluation of CT, artistic aesthetics, and code quality of projects. The obtained overall evaluations are compared with real human opinions.

5.1 Static weighted policy

We perform a simple static weighted sum of CT mastery scores, aesthetic scores, and code quality scores for the 500 projects. δ represents the weight of the CT score in the overall evaluation, μ represents the weight of the aesthetic score, and $1-\delta-\mu$ is the weight of the code quality. The results obtained using different values of δ and μ are shown in Fig. 9. It can be seen that when CT accounts for 20% of the overall evaluation, and visual aesthetics and code quality account for 65% and 15%, respectively, or when CT accounts for 30% of the evaluation, and visual aesthetics and code quality account for 50% and 20%, respectively, the overall evaluation is relatively close to the human opinions. The correlation coefficient values between the overall scores and human opinions are 0 : 65 and 0 : 62, respectively.

5.2 Dynamic weighted policy

According to the use of blocks in Scratch projects in the dataset, we combine Eq. (6) to obtain the weight of each project dynamically adjusted according to its own code quantity. When the code size is large, we focus on the CT ability embodied by the code and the code quality. When the code is less, we pay more attention to the visual presentation of the project. The results after dynamic weighting of 500 projects are shown in Fig. 10. It can be seen that part of the dynamic weighted scores is slightly lower than those of human opinions. But they are generally related and the correlation coefficient is 0.66.

5.3 Machine learning based weighted policy

We divide 500 projects into a training set and test set by 8 : 2, and use XGBoost and LightGBM for regression training, separately. The training time of XGBoost is 0.183 s. In the overall evaluation, CT accounts for 29.4%, artistic aesthetics accounts for 35.7%, and code quality accounts for 34.9%. LightGBM takes 0.026 s to train. In terms of feature importance, CT is 25.6%,

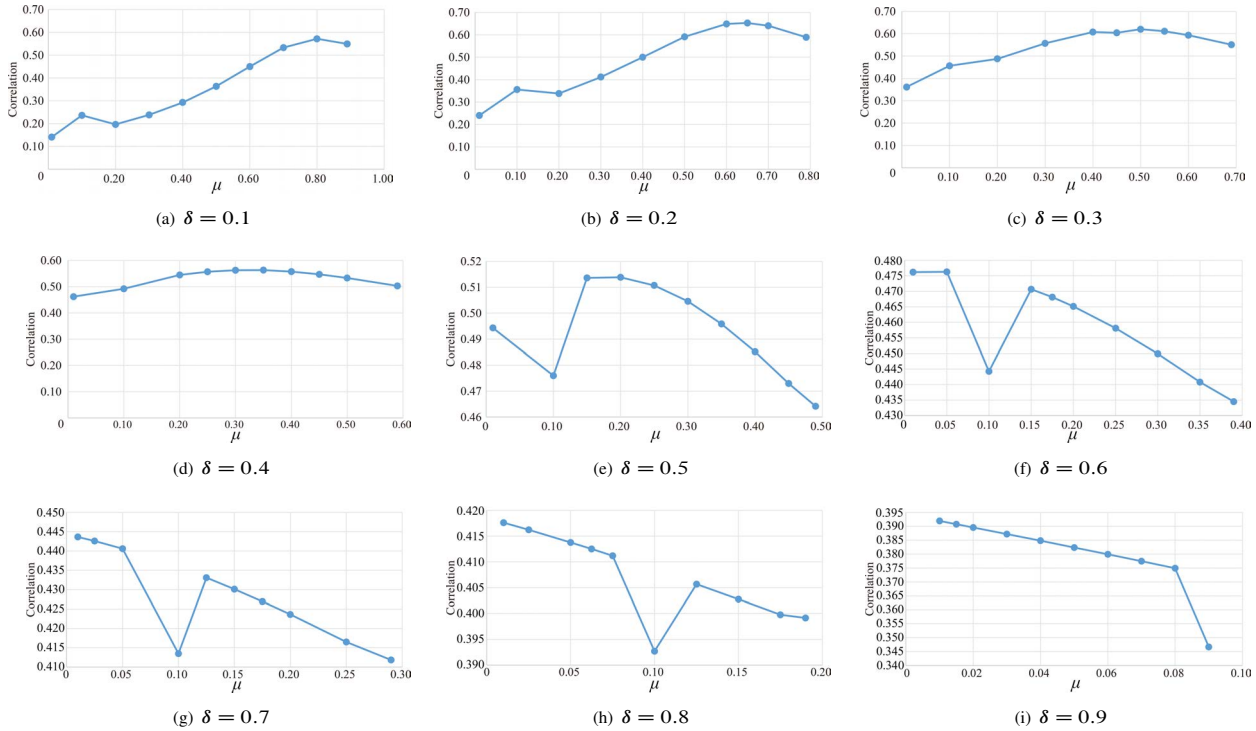


Fig. 9 Correlation of the evaluations from various perspectives with human opinions after weighted summation with different weights.

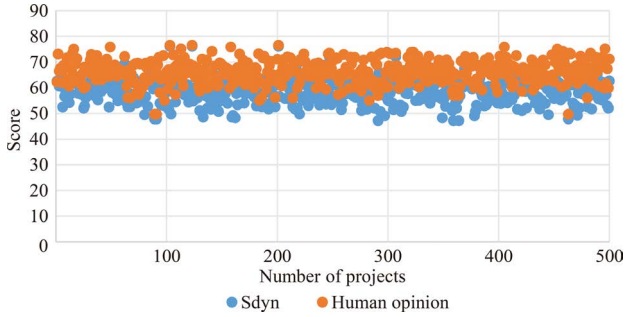


Fig. 10 Distribution of dynamically weighted scores and human opinions scores for 500 projects.

artistic aesthetics is 50.1%, and code quality is 24.3%. The prediction results of XGBoost and LightGBM on the test set are shown in Fig. 11. On the test set, the highest human truth opinion score is 76.57, the lowest is 49.68, the average is 66.20, and the median is 66.41. In the prediction results using XGBoost, the maximum value is 75.37, the minimum value is 49.95, the mean value is 65.76, and the median value is 66.07. In the overall evaluation predicted by LightGBM, the maximum is 73.51, the minimum is 54.16, the mean is 65.02, and the median is 65.59. It can be seen that the overall rating of the projects predicted by the machine learning based weighted method is roughly similar to that of the human opinion. Although the training time of LightGBM is less, the prediction results of XGBoost are generally more accurate. The correlation coefficients between XGBoost and LightGBM, and human opinions are 0.78 and 0.74, respectively.

6 Conclusion

This paper proposes a multi-view evaluation framework for Scratch. Through data analysis, we quantitatively analyze the difference and connections among the CT ability, visual aesthetics, and code quality evaluations of the projects, and compare them with human evaluations. Driven by data, three policies are proposed to integrate evaluations from different views. The weights of each component of various integration policies are determined and compared through experiments. Experiments show that our proposed multi-view evaluation method can generate overall evaluations for projects, which is close to human opinions. Such multi-dimensional comprehensive evaluation results can provide users with a more comprehensive reference.

As part of our future work, we plan to use this approach to design and implement a multi-view evaluation plugin for Scratch. The plugin can be easily integrated into any scene where project analysis is required.

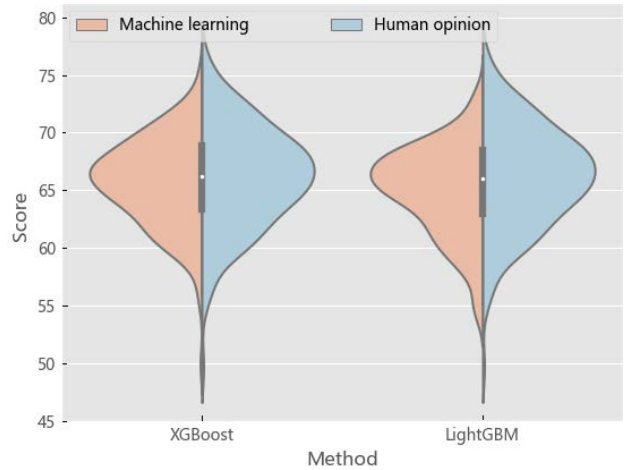


Fig. 11 Overall scores predicted with XGBoost and LightGBM versus human opinions.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (Nos. 62272052 and 62172051).

References

[1] K. Brennan and M. Resnick, New frameworks for studying and assessing the development of computational thinking, in *Proc. 2012 Ann. Meeting of the American Educational Research Association*, Vancouver, Canada, 2012, p. 25.

[2] L. Seiter and B. Foreman, Modeling the learning progressions of computational thinking of primary grade students, in *Proc. Ninth Ann. Int. ACM Conf. Int. Computing Education Research*, San Diego, CA, USA, 2013, pp. 59–66.

[3] J. M. Wing, Computational thinking and thinking about computing, *Philosoph. Trans. Roy. Soc. A Math. Phys. Eng. Sci.*, vol. 366, no. 1881, pp. 3717–3725, 2008.

[4] X. Chai, Y. Sun, H. Luo, and M. Guizani, DWES: A dynamic weighted evaluation system for scratch based on computational thinking, *IEEE Trans. Emerg. Top. Comput.*, vol. 10, no. 2, pp. 917–932, 2022.

[5] X. Lu, Z. Lin, H. Jin, J. Yang, and J. Z. Wang, RAPID: Rating pictorial aesthetics using deep learning, in *Proc. 22nd ACM Int. Conf. Multimedia*, Orlando, FL, USA, 2014, pp. 457–466.

[6] X. Lu, Z. Lin, X. Shen, R. Mech, and J. Z. Wang, Deep multi-patch aggregation network for image style, aesthetics, and quality estimation, in *Proc. 2015 IEEE Int. Conf. Computer Vision*, Santiago, Chile, 2015, pp. 990–998.

[7] S. Ma, J. Liu, and C. W. Chen, A-Lamp: Adaptive layout-aware multi-patch deep convolutional neural network for photo aesthetic assessment, in *Proc. 2017 IEEE Conf. Computer Vision and Pattern Recognition*, Honolulu, HI, USA, 2017, pp. 722–731.

[8] K. Sheng, W. Dong, C. Ma, X. Mei, F. Huang, and B. G. Hu, Attention-based multi-patch aggregation for image aesthetic assessment, in *Proc. 26th ACM Int. Conf. Multimedia*, Seoul, Republic of Korea, 2018, pp. 879–886.

[9] Z. Wang, S. Chang, F. Dolcos, D. Beck, D. Liu, and T. S. Huang, Brain-inspired deep networks for image aesthetics assessment, arXiv preprint arXiv: 1601.04155, 2016.

[10] J. Moreno-León, G. Robles, and M. Román-González, Dr. Scratch: Automatic analysis of scratch projects to assess and foster computational thinking, *RED. Revist. Edu. Distan*, vol. 46, no. 10, pp. 1–23, 2015.

[11] Z. Chang, Y. Sun, T. Y. Wu, and M. Guizani, Scratch analysis tool (SAT): A modern scratch project analysis tool based on ANTLR to assess computational thinking skills, in *Proc 14th Int. Wireless Communications & Mobile Computing Conf.* (*IWCMC*), Limassol, Cyprus, 2018, pp. 950–955.

[12] S. Kong, X. Shen, Z. Lin, R. Mech, and C. Fowlkes, Photo aesthetics ranking network with attributes and content adaptation, in *Proc. 14th European Conf. Computer Vision*, Amsterdam, The Netherlands, 2016, pp. 662–679.

[13] K. Schwarz, P. Wieschollek, and H. P. A. Lensch, Will people like your image? Learning the aesthetic space, in *Proc. 2018 IEEE Winter Conf. Applications of Computer Vision* (*WACV*), Lake Tahoe, NV, USA, 2018, pp. 2048–2057.

[14] N. Murray, L. Marchesotti, and F. Perronnin, AVA: A large-scale database for aesthetic visual analysis, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, Providence, RI, USA, 2012, pp. 2408–2415.

[15] X. Lu, Z. Lin, H. Jin, J. Yang, and J. Z. Wang, Rating image aesthetics using deep learning, *IEEE Trans. Multimedia*, vol. 17, no. 11, pp. 2021–2034, 2015.

[16] V. Hosu, B. Goldlucke, and D. Saupe, Effective aesthetics prediction with multi-level spatially pooled features, in *Proc. 2019 IEEE/CVF Conf. Computer Vision and Pattern Recognition*, Long Beach, CA, USA, 2019, pp. 9375–9383.

[17] H. Talebi and P. Milanfar, NIMA: Neural image assessment, *IEEE Trans. Image Process.*, vol. 27, no. 8, pp. 3998–4011, 2018.

[18] X. Chai, Y. Sun, H. Luo, and M. Guizani, An artistic analysis model based on sequence cartoon images for scratch, *Int. J. Intellig. Syst.*, vol. 37, no. 11, pp. 9598–9619, 2022.

[19] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, Inception-v4, inception-ResNet and the impact of residual connections on learning, in *Proc. Thirty-First AAAI Conf. Artificial Intelligence*, San Francisco, CA, USA, 2017, pp. 4278–4284.

[20] P. Techapalokul and E. Tilevich, Quality hound—an online code smell analyzer for scratch programs, in *Proc. 2017 IEEE Symp. Visual Languages and Human-Centric Computing* (*VL/HCC*), Raleigh, NC, USA, 2017, pp. 337–338.

[21] G. Fraser, U. Heuer, N. Körber, F. Obermüller, and E. Wasmeier, LitterBox: A linter for scratch programs, in *Proc. IEEE/ACM 43rd Int. Conf. Software Engineering: Software Engineering Education and Training* (*ICSE-SEET*), Madrid, Spain, 2021, pp. 183–188.

[22] F. J. M. Jr, The Kolmogorov-Smirnov test for goodness of fit, *J. Am. Stat. Assoc.*, vol. 46, no. 253, pp. 68–78, 1951.

[23] J. R. Quinlan, Induction of decision trees, *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.

[24] L. Breiman, Random forests, *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, Learning representations by back-propagating errors, *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

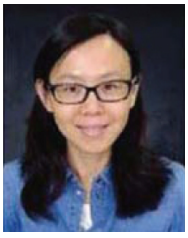
[26] J. H. Friedman, Greedy function approximation: A gradient boosting machine, *Ann. Stat.*, vol. 29, no. 5, pp. 1189–1232, 2001.

[27] T. Q. Chen and C. Guestrin, XGBoost: A scalable tree boosting system, in *Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, San Francisco, CA, USA, 2016, pp. 785–794.

[28] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Y. Liu, LightGBM: A highly efficient gradient boosting decision tree, in *Proc. 31st Int. Conf. Neural Information Processing Systems*, Long Beach, CA, USA, 2017, pp. 3149–3157.

Xiaolin Chai received the BEng degree from Beijing University of Posts and Telecommunication, China in 2017. She is currently a PhD candidate at Beijing University of Posts and Telecommunication, China. Her main research interests include data analysis, evaluation systems, and AIOps.

Yan Gao received the MEng degree from Dalian University of Technology, China in 2007. He is currently working as an associate professor at Shandong Polytechnic College. His main research fields include automatic control, pattern recognition, and intelligent agricultural machinery.



Yan Sun received the PhD degree from Beijing University of Posts and Telecommunication, China in 2007. She is currently a professor at Department of Computer Science, Beijing University of Posts and Telecommunication, China. She is also a research member of the Beijing Key Laboratory of Intelligent Telecommunication Software and Multimedia. Her research interests include big data, IoT, SDN, and AIOps.