ASEAN Journal for Science Education

Journal homepage: https://ejournal.bumipublikasinusantara.id/index.php/ajsed

Integrating Generative Artificial Intelligence (AI) and Problem-based Learning (PBL) in Programming Education: Development of Learning Activities and Computational Thinking Performance

*Nakkarin Laokhonkha, Jiraporn Chano**

Mahasarakham University, Maha Sarakham, Thailand

*Correspondence: E-mail: jiraporn.j@msu.ac.th

ABSTRACT

Generative Artificial Intelligence (AI) can assist students in programming, but generating correct code does not necessarily reflect computational thinking skills. Therefore, integrating AI with problem-based learning (PBL) may encourage students to analyze problems, evaluate solutions, and develop independent reasoning. This study developed eight programming lesson plans combining Generative AI and PBL to support Grade 11 students' computational thinking. A quasi-experimental pretest-posttest design involved 41 experimental and 38 control students studying HTML, PHP, SQL, and MySQL. Both groups completed assessments with AI assistance permitted. Mean scores increased in both groups, with greater observed improvement in the experimental group. These findings suggest potential instructional benefits, although classroom-level confounding and permitted AI assistance limit conclusions about causal effects and independent computational thinking development.

© 2027 Bumi Publikasi Nusantara

ARTICLE INFO

Article History:
Submitted/Received 07 Jun 2026
First Revised 19 Aug 2026
Accepted 24 Oct 2026
First Available online 25 Oct 2026
Publication Date 01 Dec 2027

Keyword:
Computational thinking;
Generative artificial intelligence;
Problem-based learning;
Programming education;
Secondary education.

1. INTRODUCTION

Programming education requires students to analyse problems, organise information, develop algorithms, and evaluate solutions. These processes are closely associated with computational thinking, which involves systematic problem-solving approaches applicable across scientific and technological contexts [1]. Computational thinking also includes decomposition, abstraction, algorithmic reasoning, and evaluation, making it an important competency in programming education [2]. Developing these skills is particularly relevant for secondary school students learning HTML, PHP, SQL, and MySQL, where understanding logical structures is essential for designing functional programs.

Problem-based learning (PBL) provides an instructional approach that encourages students to investigate problems, develop solutions, and reflect on their reasoning [3]. Previous research has demonstrated the potential of problem-oriented programming instruction to support computational thinking, although improvements may vary across specific skill dimensions [4]. Technology-supported learning activities have also been investigated to promote computational thinking among secondary school students [5]. Bibliometric research further highlights the continuing interest in computational thinking within educational research [6].

Generative Artificial Intelligence (AI) offers additional opportunities for programming education by providing explanations, coding examples, and feedback. However, AI-generated solutions may allow students to complete programming tasks without fully understanding the underlying processes [7]. Previous research on AI code generators has demonstrated benefits for novice programming performance while raising questions about independent learning and knowledge transfer. Generative AI-based learning approaches have also been explored for supporting educational activities and student literacy [8]. AI should be integrated into instructional activities that require students to analyse, verify, and explain generated solutions rather than simply reproduce them.

Effective integration of AI into PBL requires alignment among learning objectives, instructional activities, and assessment [9]. Previous studies have also examined PBL as an approach for improving motivation in science education [10]. Nevertheless, concerns regarding AI dependency and student creativity highlight the importance of maintaining active learner participation [11]. Responsible educational AI integration should therefore prioritize learner agency, instructional suitability, and critical evaluation of AI-generated content [12].

Despite growing interest in computational thinking, PBL, and Generative AI, their integration into secondary school programming activities requires further classroom-based investigation. In particular, the development of structured learning activities combining programming problems, AI-assisted scaffolding, and computational thinking assessment remains an important instructional consideration. This study aims to develop programming learning activities integrating Generative AI and PBL for Grade 11 students, examine changes in computational thinking performance before and after instruction, and descriptively compare the observed performance of students receiving integrated instruction with those receiving conventional instruction. The study focuses on programming activities involving HTML, PHP, SQL, and MySQL under common assessment rules permitting AI assistance.

2. LITERATURE REVIEW

2.1. Computational Thinking in Programming Education

Computational thinking involves identifying problems, organizing information, designing procedures, and evaluating solutions. In programming education, these processes can be represented through decomposition, pattern recognition, abstraction, algorithm design, and debugging. Computational thinking frameworks also emphasize the importance of examining learners' computational concepts, practices, and perspectives rather than relying exclusively on completed programming products [13].

2.2. Problem-Based Learning and Instructional Design

PBL encourages students to explore authentic problems, identify knowledge requirements, investigate alternatives, and evaluate proposed solutions [3]. In programming education, these activities can involve analysing user requirements, designing databases, validating inputs, and debugging programs. Problem-oriented programming activities can produce different outcomes across computational thinking components [4]. Therefore, the instructional activities in this study were designed to connect programming tasks with identifiable problem-solving processes. Constructive alignment was applied to ensure consistency between intended learning outcomes, instructional activities, and assessment evidence [9].

2.3. Generative AI as Instructional Scaffolding

Generative AI can support programming education by providing explanations, examples, and suggestions during problem-solving activities. However, the availability of AI-generated code requires students to evaluate whether suggested solutions are accurate and appropriate for specific programming problems [7]. Previous studies have demonstrated the potential of AI code generators to support novice learners while highlighting the need to distinguish assisted task performance from independent understanding. Research on educational uses of ChatGPT has also highlighted the importance of teacher guidance and appropriate integration into classroom activities [13].

3. METHODOLOGY

3.1. Research Design and Participants

This study employed a quasi-experimental pretest–posttest design involving two intact Grade 11 classes at a secondary school in Maha Sarakham Province, Thailand, during the first semester of the 2026 academic year (B.E. 2569). The participants were enrolled in a Design and Technology course covering HTML, PHP, SQL, and MySQL. The course consisted of two instructional periods per week and carried one academic credit. The population comprised 158 students from four Grade 11 classes taught by the researcher. Two classes were selected through cluster random sampling. Class 5/1, consisting of 41 students, served as the experimental group, while Class 5/2, consisting of 38 students, served as the control group. Selection of intact classes was distinguished from assignment to instructional conditions because random allocation of the selected classes was not documented. Consequently, the study was classified as quasi-experimental rather than a cluster-randomised trial. Classroom-level differences were considered when interpreting the results [14]. **Table 1** summarizes the study groups. All 79 students from the selected classes completed the pretest and posttest assessments, providing complete paired observations for analysis.

Table 1. Course classes and participant distribution.

CLASS	NUMBER OF STUDENTS	ROLE
5/1	41	Experimental group: PBL with Generative AI
5/2	38	Control group: conventional instruction
5/3 and 5/4	79	Not included in the comparison
Total	158	79 students included in the analysis

3.2. Development of Programming Learning Activities

The instructional activities were developed by integrating Generative AI with PBL to support computational thinking in programming education. Eight lesson plans were prepared, covering four problem-solving cycles related to school activity records, data organisation, input validation, and database retrieval. Each lesson plan comprised two 50-minute periods, providing a total of 800 instructional minutes across eight teaching weeks. Pretest and posttest administration were conducted separately. The activities followed 6 PBL stages: problem presentation, problem analysis, investigation, implementation, reflection, and presentation with evaluation. Students initially analysed programming problems and developed possible solutions before using ChatGPT for additional explanations or programming suggestions. Students were expected to evaluate AI-generated responses, verify their accuracy, and explain their programming decisions. An AI-use recording format was designed to document students' initial attempts, prompts, generated responses, decisions, and verification evidence; however, adherence to the protocol was not quantitatively evaluated.

The experimental group received instruction integrating PBL with structured AI scaffolding, whereas the control group received conventional instruction consisting of explanation, demonstration, and programming practice. Both groups were taught by the same researcher using comparable programming content, learning objectives, and instructional time. AI and internet access were permitted under the same AI Usage Protocol during assessments. Therefore, the instructional comparison primarily concerned the structured integration of AI within PBL rather than the availability of AI alone.

3.3. Computational Thinking Assessment

Computational thinking performance was assessed using a 30-point instrument consisting of two programming problem scenarios. Each scenario evaluated five dimensions: decomposition, pattern recognition, abstraction, algorithm design, and debugging. Each dimension was scored from 0 to 3, producing a maximum score of 30 points across the two scenarios. The same assessment questions were administered during the pretest and posttest, with their order changed for the posttest. Both groups were permitted to use AI and internet resources under the same protocol. Students were instructed to attempt responses independently before consulting AI for checking or extending their understanding. Consequently, assessment scores represented computational thinking performance under conditions permitting AI assistance rather than exclusively unassisted performance. We evaluated student responses using a rubric ranging from absent or irrelevant evidence to coherent and justified reasoning. The total score was calculated from ten dimension ratings. The study did not include independent scoring, dimension-specific statistical analysis, or formal external validation of the assessment instrument. Accordingly, the reported outcomes were interpreted as total task-performance scores.

3.4. Data Collection and Analysis

Data collection involved administering the computational thinking assessment before and after the instructional intervention. Students' scores were checked for completeness and consistency with the 0-30 scoring range. Descriptive statistics, including means and standard deviations, were calculated for pretest scores, posttest scores, and individual gain scores. Gain scores were determined using $G_i = X_{\text{post},i} - X_{\text{pre},i}$, where G_i represents the gain score of student i, $X_{post,i}$ represents the posttest score, and $X_{pre,i}$ represents the pretest score. Shapiro-Wilk tests were used to screen gain-score distributions, with reported p-values of 0.129 and 0.055 for the experimental and control classes, respectively. Within-class changes were evaluated using two-sided paired-samples t-tests, with 95% confidence intervals calculated for mean gains. Paired-samples analysis was applied to examine differences between repeated measurements for the same participants. Effect sizes were calculated to describe the magnitude of within-class changes, following established recommendations for reporting standardised mean differences [15]. Calculations used unrounded scores, while reported statistics were rounded for presentation. Between-class comparisons were conducted descriptively using differences in pretest means, posttest means, and mean gains. Because each instructional condition involved only one classroom, classroom characteristics could not be separated from instructional effects. Therefore, no between-condition significance test was reported, and the observed differences were not interpreted as evidence of a causal treatment effect.

4. RESULTS AND DISCUSSION

4.1. Development of Programming Learning Activities Integrating Generative AI and PBL

Table 2 presents the sequence of the eight developed programming lesson plans, including their learning focus, expected evidence, and instructional duration. The sequence was designed to progress from understanding programming problems to developing and evaluating complete solutions. The first two plans emphasized problem decomposition and data organisation, whereas Plans 3-6 addressed algorithm development, input validation, and debugging. Plans 7 and 8 required students to integrate their understanding into a more complete programming solution. This progression was consistent with constructive alignment, which emphasizes relationships among intended learning outcomes, instructional activities, and assessment evidence [9]. Problem-oriented programming instruction may support different components of computational thinking, highlighting the importance of explicitly connecting learning activities with the intended competencies [4].

Table 2. Sequence of the eight developed programming lesson plans.

PLAN	PROBLEM AND LEARNING FOCUS	EVIDENCE TO BE ELICITED	TIME (MIN)
1	Duplicate school activity records; analyze requirements and divide tasks	Problem map and knowledge needs	100
2	Organize activity data into tables and relationships	Data model and sample-case checks	100
3	Specify conditions for valid data entry	Pseudocode and predicted test outcomes	100
4	Implement and correct data entry and updating	Program revisions and debugging log	100
5	Determine which records a report should include	Selection rules and expected results	100
6	Correct queries that produce inaccurate results	Revised queries and test evidence	100
7	Design a small system under new requirements	System design and test plan	100
8	Test, explain, and refine the completed solution	Artifact, explanation, and limitations	100
Total			**800**

Generative AI was positioned as a supporting resource rather than a substitute for student reasoning. Students were expected to attempt solutions before requesting assistance, evaluate AI-generated suggestions, and explain their programming decisions. Students may produce functional programs without fully understanding the underlying computational processes [7]. However, because adherence to the designed AI-use protocol was not quantitatively evaluated, the extent to which students followed these procedures cannot be established from the available data.

4.2. Alignment Between Learning Activities and Computational Thinking Dimensions

Table 3 summarizes the relationships between the computational thinking dimensions, designed learning opportunities, and expected assessment evidence. The alignment demonstrated how programming problems were designed to elicit evidence of computational thinking. For example, decomposition was addressed when students separated a programming task into data input, validation, storage, and reporting components. Pattern recognition was incorporated through comparisons of recurring records and conditions, while abstraction required students to identify essential information and exclude unnecessary details. Algorithm design was represented through the development of programming procedures and decision structures. Debugging required students to compare expected and observed outputs, identify possible causes of errors, modify solutions, and conduct additional tests. These activities reflected computational thinking as a combination of concepts and practices rather than solely the ability to produce a completed programming artifact [12]. The integration of Generative AI provided opportunities for students to obtain explanations and programming examples during these processes. However, AI-generated responses required critical evaluation because automatically generated solutions do not necessarily demonstrate independent computational understanding. The present framework therefore emphasised the relationship between learning activities and observable reasoning evidence. The alignment presented in **Table 3** represents the instructional design of the developed activities. It does not constitute independent expert validation or empirical evidence that each computational thinking dimension improved. Dimension-specific analysis would be required to determine whether particular skills benefited more than others.

Table 3. Alignment of computational thinking with task evidence.

DIMENSION	DESIGNED LEARNING OPPORTUNITY	EVIDENCE FOR ASSESSMENT
Decomposition	Separate input, validation, storage, and reporting	Identify subtasks and explain dependencies
Pattern recognition	Compare recurring records and conditions	State a shared rule and examine a new case
Abstraction	Select required fields and relationships	Represent necessary data and justify exclusions
Algorithm design	Specify operations, decisions, and outputs	Give a traceable procedure with relevant conditions
Debugging	Compare observed and predicted outputs	Locate a cause, revise, and retest the solution

4.3. Computational Thinking Performance Before and After Instruction

Table 4 presents the descriptive statistics for the experimental and control groups. The experimental group's mean score increased from 19.10 to 24.68 points, corresponding to an observed mean gain of 5.59 points. The control group's mean score increased from 19.00 to 21.29 points, corresponding to an observed mean gain of 2.29 points. Both groups therefore demonstrated improvement in their computational thinking assessment performance. All 41

students in the experimental group obtained higher posttest scores than pretest scores. In the control group, 33 students improved, while five students recorded unchanged scores. No score decreases were observed in either group. Additionally, four experimental-group students achieved the maximum posttest score of 30, compared with none in the control group. The experimental class demonstrated greater observed improvement during the instructional period. One possible explanation is that structured programming problems encouraged students to organise their reasoning and evaluate solutions, while Generative AI provided additional explanations and feedback. Problem-oriented programming instruction can support computational thinking development, although the outcomes may differ across specific dimensions [4]. However, the observed improvement cannot be attributed exclusively to the combination of PBL and Generative AI. Students in both groups were permitted to use AI and internet resources during assessments, and actual AI usage was not measured. Consequently, the reported scores represent performance under assisted assessment conditions rather than independent computational thinking performance.

Table 4. Performance on computational thinking tasks by group.

GROUP	*n*	PRETEST MEAN (SD)	POSTTEST MEAN (SD)	GAIN MEAN (SD)
Experimental (5/1)	41	19.10 (3.35)	24.68 (3.78)	5.59 (1.72)
Control (5/2)	38	19.00 (3.65)	21.29 (4.25)	2.29 (1.54)

Note: SD = standard deviation; n = number of paired records. The maximum possible score was 30. Gain scores were calculated as posttest minus pretest scores. Calculations used unrounded scores; therefore, subtracting the displayed rounded means may not reproduce the reported gains exactly.

4.4. Statistical Analysis of Pretest and Posttest Performance

Paired-samples t-tests were conducted to examine whether computational thinking assessment scores changed significantly within each class. The results are presented in **Table 5**. The paired-samples analysis indicated statistically significant improvements within both classes. For the experimental group, the mean gain was 5.59 points, with a 95% confidence interval ranging from 5.04 to 6.13 points. The corresponding test statistic was $t(40) = 20.83$, with $p < 0.001$. For the control group, the mean gain was 2.29 points, with a 95% confidence interval ranging from 1.78 to 2.80 points. The corresponding test statistic was $t(37) = 9.16$, with $p < 0.001$. Both classes demonstrated positive changes in assessment performance under the stated assumptions. The calculated within-class effect sizes were $d_z = 3.25$ for the experimental group and $d_z = 1.49$ for the control group. The corresponding d_{av} values were 1.57 and 0.58, respectively. These alternative standardisations provide complementary descriptions of the magnitude of change and should be interpreted according to the underlying variability of the repeated measurements [15].

Table 5. Paired-samples t-tests of pretest–posttest change within each class.

GROUP	95% CI for Mean Gain	t(df)	p-value	Cohen's d_z
Experimental (5/1)	[5.04, 6.13]	20.83 (40)	< 0.001	3.25
Control (5/2)	[1.78, 2.80]	9.16 (37)	< 0.001	1.49

Note: CI = confidence interval; df = degrees of freedom. Tests were two-sided. Cohen's d_{av} was 1.57 for the experimental group and 0.58 for the control group, complementing d_z, whose standardizer reflects the high pretest–posttest correlations (r = 0.89 and 0.94, respectively). Statistical inference assumes independent student gains within each class and does not establish a causal treatment effect.

Although the experimental group demonstrated a larger observed gain and within-class effect size, comparing the statistical significance of two separate paired-samples tests does not establish a statistically significant difference between instructional conditions. Furthermore, because each condition involved only one classroom, the effects of classroom characteristics and instructional approaches could not be separated.

4.5. Comparison of Experimental and Control Groups

Table 6 presents differences in their pretest means, posttest means, and mean gains. The difference between the groups' initial mean scores was 0.10 points, suggesting similar average starting scores on the administered assessment. After instruction, the experimental group's mean exceeded the control group's mean by 3.39 points. The difference in mean gains was 3.30 points. The class receiving integrated Generative AI and PBL instruction experienced a greater observed improvement than the class receiving conventional instruction. The instructional activities may have provided opportunities for students to engage more systematically in analyzing programming problems, evaluating solutions, and reflecting on computational decisions. Such processes are consistent with the educational principles underlying PBL [3]. Generative AI may also have supported students by providing explanations and programming suggestions during learning activities. Benefits associated with AI-supported programming performance do not necessarily translate into equivalent improvements on independent assessments conducted without AI assistance. The present study cannot determine whether the observed gain difference resulted from PBL, structured AI scaffolding, classroom characteristics, or interactions among these factors. Although the two classes had similar initial mean scores, this similarity does not establish equivalence in all relevant characteristics. The 3.30-point difference should be interpreted as a descriptive classroom-level observation rather than an independently established causal instruction.

Table 6. Observed differences between classroom means.

MEASURE	EXPERIMENTAL CLASS MEAN	CONTROL CLASS MEAN	DIFFERENCE (E-C)
Pretest	19.10	19.00	0.10
Posttest	24.68	21.29	3.39
Gain	5.59	2.29	3.30

Note: E-C = experimental mean minus control mean. Differences were calculated using unrounded values. Both groups followed the same AI access rules. The comparison is descriptive because only one classroom represented each instructional condition.

4.6. Educational Implications and Study Limitations

The findings offer preliminary implications for integrating Generative AI:

(i) The developed lesson plans illustrate how AI can be incorporated into structured problem-solving activities without positioning generated code as the sole indicator of successful learning. Requiring students to analyze problems, explain programming decisions, and verify AI-generated responses may provide opportunities to engage with computational thinking processes.

(ii) The alignment between the learning activities and assessment criteria provides an instructional framework connecting programming tasks with decomposition, pattern recognition, abstraction, algorithm design, and debugging. This approach is consistent with the view that computational thinking should be assessed through learners' reasoning and practices rather than only their completed products [12].

(iii) The results emphasise the importance of distinguishing assessment performance under permitted AI assistance from independently demonstrated knowledge. Responsible integration of Generative AI requires consideration of student agency, appropriate pedagogical use, and critical examination of AI-generated information. Similar concerns regarding student dependency and the role of feedback have been discussed in research on ChatGPT-supported educational activities [11].

Several limitations should be considered. Only one classroom was included in each instructional condition, preventing the separation of classroom differences from possible instructional effects. PBL and structured AI scaffolding were introduced simultaneously, so their individual contributions could not be determined. The same researcher developed the instructional activities, taught both classes, and scored the assessments, while independent or masked scoring was not documented. The same assessment questions were administered before and after instruction, with their order changed, creating the possibility of practice effects. Four students in the experimental group reached the maximum posttest score, suggesting potential ceiling effects. The assessment instrument did not undergo documented independent validation or second-rater reliability assessment, and dimension-level performance was not reported. Both groups were permitted to use AI during assessment, while actual AI usage and compliance with the AI-use protocol were not measured. Therefore, conclusions about independent computational thinking development remain limited. Future research should include multiple classrooms per instructional condition, documented group allocation, independently evaluated assessment instruments, and statistical analyses accounting for classroom clustering. Additional assessments conducted without AI assistance, preferably using new programming problems, would help determine whether improvements transfer to independent computational thinking performance. Detailed AI-use records and dimension-level analyses could also clarify how students engage with different computational thinking processes.

5. CONCLUSION

This study developed eight programming lesson plans integrating Generative AI and problem-based learning to support computational thinking among Grade 11 students. Both classes improved on the 30-point assessment, with larger observed gains in the experimental class (5.59 points) than in the control class (2.29 points). The activities connected programming problems, guided AI use, and five computational-thinking dimensions. Nevertheless, one classroom per condition, repeated assessment questions, researcher scoring, and permitted AI assistance limit interpretation. The findings support further evaluation of integrated instruction using multiple classrooms, independent assessment, documented AI use, and new tasks completed without assistance to examine transferable learning outcomes.

6. ACKNOWLEDGMENTS

We thank the participating school, teachers, and Grade 11 students for their cooperation during the implementation of the programming learning activities and assessments. We also acknowledge the Faculty of Education, Mahasarakham University.

7. AUTHORS' NOTE

The authors declare that there is no conflict of interest regarding the publication of this article. The authors confirmed that the paper is free from plagiarism.

8. REFERENCES

[1] Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35.

[2] Shute, V. J., Sun, C., and Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142-158.

[3] Hmelo-Silver, C. E. (2004). Problem-based learning: What and how do students learn?. *Educational Psychology Review*, *16*(3), 235-266.

[4] Bai, H., Wang, X., and Zhao, L. (2021). Effects of the problem-oriented learning model on middle school students' computational thinking skills in a Python course. *Frontiers in Psychology*, *12*, 771221.

[5] Reskianissa, A., Sakti, A. W., and Azizah, N. N. (2022). TikTok platform to train middle school students' computational thinking skills in distance learning. *ASEAN Journal of Educational Research and Technology*, *1*(1), 79-86.

[6] Abidin, Z., Herman, T., Wahyudin, W., and Farokhah, L. (2025). Bibliometric analysis using VOSviewer with Publish or Perish of computational thinking and mathematical thinking in elementary school. *ASEAN Journal for Science Education*, *4*(1), 7-16.

[7] Becker, B. A., Denny, P., Finnie-Ansley, J., Luxton-Reilly, A., Prather, J., and Santos, E. A. (2023). Programming is hard-or at least it used to be: Educational opportunities and challenges of AI code generation. *Proceedings of the 54th ACM Technical Symposium on Computer Science Education, 1*, 500-506.

[8] Nurjamin, A., Nurjamin, L. R., Fajriah, Y. N., Nurjamin, A. K., and Firdaus, H. A. (2026). Integrating generative artificial intelligence (AI)-based multimodal learning in education to enhance literacy aligned with Sustainable Development Goals (SDGs). *ASEAN Journal of Educational Research and Technology*, *5*(1), 71–88.

[9] Biggs, J. (1996). Enhancing teaching through constructive alignment. *Higher Education*, *32*(3), 347-364.

[10] Arifiani, I., Hamidah, L. N., and Rahmawan, S. (2025). Problem based learning (PBL) learning model for increasing learning motivation in chemistry subject: Literature review with bibliometric analysis. *ASEAN Journal for Science Education*, *4*(1), 17-30.

[11] Katel, K. (2026). Artificial intelligence (AI) ChatGPT in academic and science writing education: Student dependency, creativity, and the mediating role of feedback in English as a second language and English as a foreign language contexts. *ASEAN Journal for Science Education*, *5*(2), 155-166.

[12] Brennan, K., and Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *Proceedings of the 2012 annual meeting of the American Educational Research Association*, *1*(25), 13-17.

[13] Thapa, M. B., and Shrestha, K. (2026). Empowering teachers through ChatGPT: Perceptions and experiences in English as a foreign or second language (EFL/ESL) classrooms as a form of community service. *ASEAN Journal of Community Service and Education*, *5*(1), 21-32.

[14] Campbell, M. K., Piaggio, G., Elbourne, D. R., and Altman, D. G. (2012). Consort 2010 statement: extension to cluster randomised trials. *Bmj*, *345*, e5661.

[15] Lakens, D. (2013). Calculating and reporting effect sizes to facilitate cumulative science: a practical primer for t-tests and ANOVAs. *Frontiers in Psychology*, *4*, 863.