The Electronic Journal of Mathematics and Technology, Volume 20, Number 2, ISSN 1933-2823

Adaptive Flick-Input TeX Interface: Dynamic Scaling and LLM-Driven Step-by-Step Learning on Mobile Devices

Takuya Kitamoto
kitamoto@yamaguchi-u.ac.jp
Faculty of Education
Yamaguchi University
753-8513
Japan

Hisashi Usui
usui@gunma-ct.ac.jp
Gunma College
National Institute of Technology
371-8530
Japan

Naoki Hamaguchi
hama@nagano-nct.ac.jp
Nagano College
National Institute of Technology
381-8550
Japan

Masataka Kaneko
masataka.kaneko@phar.toho-u.ac.jp
Faculty of Pharmaceutical Sciences
Toho University
274-8510
Japan

Takeo Noda
noda@phar.toho-u.ac.jp
Faculty of Science
Toho University
274-8510
Japan

Abstract

Although flick-based input methods have reduced barriers to mathematical formula entry on mobile devices, limited smartphone screen space remains a critical bottleneck for advanced STEM learning. This limitation becomes especially apparent when the interface must accommodate both formula input and either lengthy AI-generated explanations or interactive diagrams.

This paper presents an extended version of the Flick-Input TeX System featuring Dynamic Interface Scaling and modular integration with Large Language Models (LLMs). The proposed system provides a flexible architecture in which the number and size of flick-input keys can be reconfigured in real time through JavaScript function calls or automatic page transitions. This capability allows the interface to switch dynamically to Compact Mode, thereby reclaiming screen space for step-by-step mathematical derivations and larger interactive diagrams without excessive scrolling. By combining human-centered flick gestures with AI-driven support in a space-efficient environment, the proposed system provides a robust platform for advanced STEM learning on mobile devices.

1 Introduction

Recent advances in flick-based input methods for smartphones have significantly improved the speed and usability of mathematical entry compared with traditional on-screen keyboards ([1]-[4]). As

digital learning environments continue to evolve, TeX remains the de facto standard for the precise representation of complex formulas, particularly when equations are submitted to large language models (LLMs) such as ChatGPT. Despite its usefulness, however, TeX imposes a high cognitive load on learners, who must memorize commands such as `\sqrt` and `\frac` and use them within a strict syntactic structure.

To reduce these barriers, previous research [5] introduced a flick-input TeX interface that allows users to select macros through intuitive directional swipes. This approach eliminates the need for command memorization and enables even novice users to enter mathematical expressions efficiently. In classroom trials with undergraduate students, more than 90% of participants completed formula-entry tasks successfully, although 70% had no prior experience with TeX.

Nevertheless, mobile STEM learning continues to be hindered by the three walls of input: the burden of memorizing commands, the repetitive tapping required on small screens, and the excessive screen space occupied by software keyboards. Although the existing system was built on e-learning and mathematical frameworks described in [5]-[9], its fixed layout often made it difficult to balance the input area with the display space required for detailed AI-generated explanations or intricate diagrams.

This paper presents an advanced extension of the flick-input system to address these limitations. The present journal article is a substantial extension of our earlier paper presented at the Asian Technology Conference in Mathematics (ATCM) [5]. Whereas the ATCM paper introduced a flick-based TeX input interface designed mainly to reduce the difficulty of mathematical expression entry on mobile devices, the current work advances that foundation in several important ways. Specifically, it adds dynamic interface scaling, dual layout modes for more efficient use of limited screen space, modular and pluggable LLM integration, and space-optimization mechanisms for displaying both step-by-step AI-generated explanations and interactive diagrams. In this sense, the journal version moves beyond a proof-of-concept input tool and presents a more comprehensive adaptive learning platform for mobile mathematics education.

The main contributions of this work are as follows:

- **Dynamic Interface Scaling**:The system allows dynamic modification of key layouts and key sizes through page transitions or JavaScript function calls, enabling more flexible use of limited screen space.
- **Seamless LLM Integration**: The architecture has been extended to incorporate generative AI in a modular manner, enabling the system to provide structured, step-by-step explanations rather than only final answers.

Together, these extensions move the system beyond simple input efficiency toward a comprehensive mobile learning platform that integrates intuitive human input with AI-supported explanation.

2 Related Work and Foundation

This section reviews the original Flick-Input TeX Interface and the software frameworks on which the present system is built, primarily based on the work reported in [5]-[9].

2.1 The Previous Flick-Input TeX Interface

The primary objective of the original system was to reduce barriers to mathematical formula entry on mobile devices. By grouping related mathematical symbols into single buttons and mapping directional swipes to specific macros, the system enabled users to enter complex equations with far fewer touch operations than conventional on-screen keyboards.

2.2 Core Software Architecture

The technical foundation of the system rests on the KeTCindy.JS and KeTMath frameworks. A key implementation step was the transition from standard HTML5 buttons to a coordinate-based tracking mechanism. Because standard buttons have limited capability for rendering mathematical notation clearly, the system uses a JavaScript–CindyScript bridge to track touch positions in real time.

2.3 Integration with AI and Educational Results

The effectiveness of the interface was examined in classroom trials, in which approximately 70% of the students were using TeX for the first time. To support AI-integrated learning, the system allows users to confirm their input through immediate rendering by the KaTeX engine and provides a seamless connection between mathematical expressions and LLM-based queries.

3 Implementation of Dynamic Interface Scaling

Building on the foundational architecture, this study introduces an enhanced system that supports dynamic modification of keyboard scale and layout.

3.1 Configuration Triggers

The number and size of keys can be adjusted in real time through two main triggers:

(a) **Page Transitions**: Keyboard configurations can be predefined for specific pages so that the interface adapts automatically to the lesson context.

(b) **JavaScript Function Calls**: Specific functions can be invoked to change key dimensions or the total number of keys during operation.

As a result, the interface can switch between Standard Mode, which is optimized for rapid entry with larger touch regions, and Compact Mode, which maximizes the visible display area.

3.2 Key Customization

The flick-input component is implemented in JavaScript, which allows extensive customization of both key types and the total number of keys. As discussed in previous studies, this flexibility enables the interface to adapt to a wide range of mathematical requirements beyond a standard layout. The JavaScript structure used for these configurations is shown in Code 1.

Code 1: JavaScript code for key customization

```
// Compact Mode Configuration
suunomi=[
[["1 2","3 4 5"],["1","2","3","4","5"],["1","2","3","4","5"]],
[["6 7","8 9 0"],["6","7","8","9","0"],["6","7","8","9","0"]],
[["$=\ +\ -$","$*\ \ /$"],["$=$","$+$","$*$","$-$","$/$
   "],["=","+","\ast","-","tx(/)"]],
[["$x\ y\ z$","$s\ t$"],["$x$","$y$","$z$","$s$","$t$"],["x","y
   ","z","s","t"]],
[["( . )","$\times \ \div$"],[".","(","$\times$",")","$\div$
   "],[".","(","\times",")","\div"]],
[["",""],["","","","","",""],["","","","","",""]]
];

// Standard Mode Configuration
hairetu1=[
[["sin log","cos tan ln"],["sin","cos","tan","log","ln"],["sin(?)
   ","cos(?)","tan(?)","log(?)","ln(?)"]],
[["1",".^/_"],["1",".","^","/","_"],["1",".","^","/","_"]],
[["2","abc #"],["2","a","b","c","#"],["2","a","b","c","\#"]],
[["3","def %"],["3","d","e","f","%"],["3","d","e","f","\%"]],
[["Undo","Pull Push"],["Undo","","Pull","","Push"],["Delete()","
   Delete()","Pull()","Delete()","Push()"]],
[["fr $\sqrt{\phantom{a}}$_^","",],["fr","tfr","$\sqrt{\phantom{a}}
   $","^","_"],["fr(?,)","tfr(?,)","sq(?)","^(?)","_(?)"]],
[["4", "ghi ("],["4","g","h","i","("],["4","g","h","i","("]],
[["5","jkl &"],["5","j","k","l","&"],["5","j","k","l","\&"]],
[["6","mno )"],["6","m","n","o",")"],["6","m","n","o",")"]],
[["( <","",],["(","{","[","<","$\leq$"],["(","\{","[","<","(leq)
   "]],
[["$\sum \int \ \frac{dy}{dx}$","$[\phantom{aa}]_\cdot^\cdot \ \ \
   frac{\partial z}{\partial x}$"],
["$\int$","$\sum$","$[\phantom{a}]_\cdot^\cdot$","$\frac{dy}{dx}$
   ","$\frac{\partial z}{\partial x}$"],
["int(?,,,)","sum(?,,)","br(?,,)","diff(?,)","par(?,)"]],
[["7","pqrs"],["7","p","q","r","s"],["7","p","q","r","s"]],
[["8","tuv ="],["8","t","u","v","="],["8","t","u","v","="]],
[["9","wxyz"],["9","w","x","y","z"],["9","w","x","y","z"]],
[[") >","",],[")","}","]",">","$\geq$"],[")","\}","]",">","(geq)
   "]],
[["lim mat $\cdot$","det case $\cdots$"],["lim","mat","det","case
   ","$\cdot$"],["lim(?,,)","mat(?,;,)","det(?,;,)","case(?,;,)
   ","dot()"]],
[["’ $\pm$ text","space"],["$\pm$","$\mp$","’","text","space
   "],["(pm)","(mp)","’","tx(?)","(sp)"]],
[["0","$+ \ - \ \times \ \div$"],["0","-","+","$\times$","$\div$
   "],["0","-","+","\times ","\div"]],
[["$\pi\ \infty$ !","| \"],["$\pi$","$\infty$","!","|","\"],["pi
   ","(inf)","!","|","\"]],
[[",",";  .  :  /"],[",",";",":",".","/"],[",",";",":",".","/"]],
[["",""],["","","","","",""],["","","","","",""]]
];
```

In Code 1, the variable `suunomi` defines the key layout for Compact Mode, whereas `hairetu1` defines the layout for Standard Mode. These flick-input keys are represented as structured JavaScript arrays, and each element corresponds to the labels and output macros assigned to a specific key. By modifying these arrays or adding new elements, developers can create specialized key sets tailored to different mathematical topics and instructional needs.

4 Integration of LLM Components

The Large Language Model (LLM) functionality is implemented as a pluggable component. When the user clicks the "Add LLM" button shown in Fig. 1 (left), the LLM component appears, as shown in Fig. 1 (center). Its position and size can be adjusted freely by dragging either the title bar or the light-blue square handle at the bottom-right corner.

Fig. 1: Interface sequence showing the addition and use of an LLM component: add button (left), component immediately after activation (center), and component after receiving a response (right).

In Fig. 1 (center), a query can be sent to the LLM by entering a prompt in the text field labeled "Input LLM prompt" and clicking the "Query" button. Fig. 1 (right) shows an example response generated after the prompt "Hello" is submitted.

These query operations can also be executed through JavaScript programs. Figure 2 (left) shows a learning material that uses the LLM component.

In this material, the user enters a function in the blank field below the "To Portrait" button, using flick input if needed. When the "Differentiate" button is clicked, the system provides not only the final answer but also a step-by-step explanation of the differentiation process.

Code 2 shows the JavaScript program executed when the "Differentiate" button is clicked.

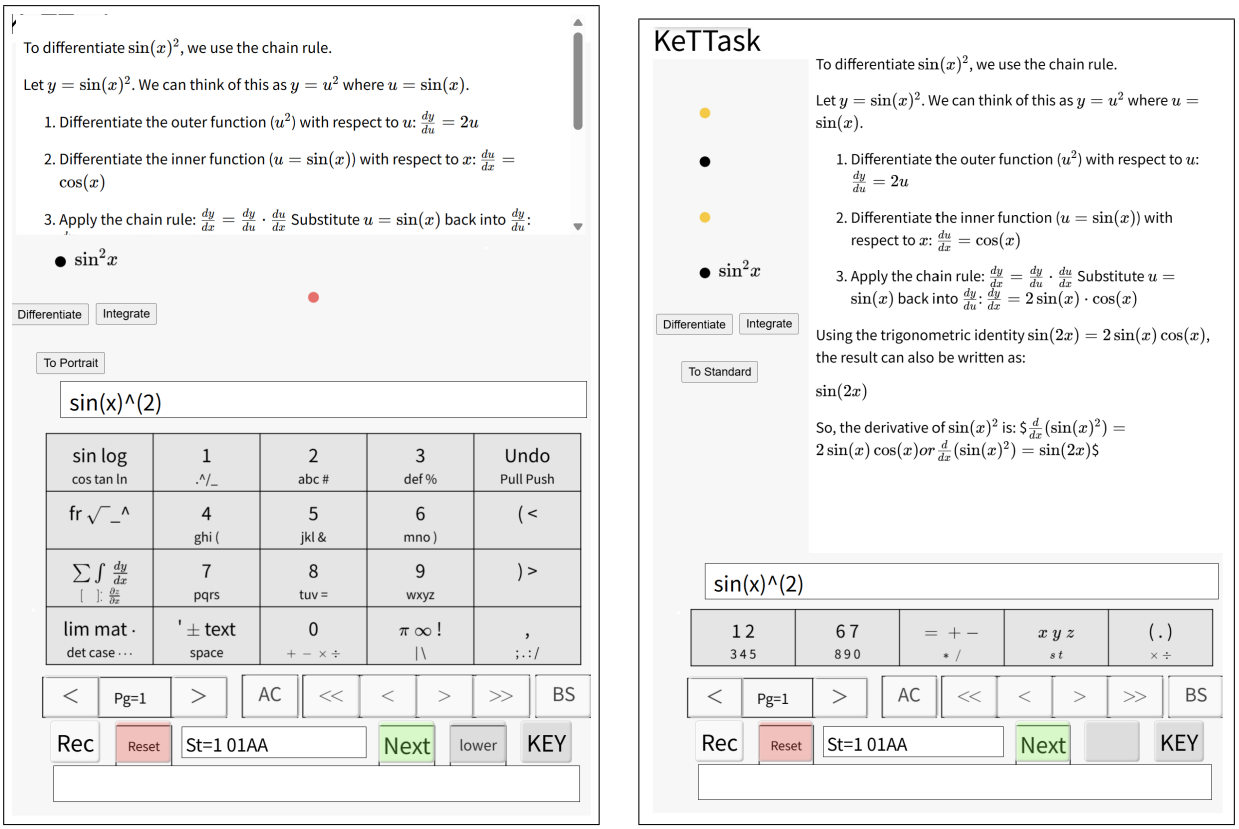

Fig. 2: Learning material for function differentiation with the LLM display in Standard Mode (left) and Portrait Mode (right).

Code 2: Code for Differentiation

```
1 var fx = get_ketmath_input_raw();
2 var prompt = "Differentiate " + fx + ". Format math with $ or $$
     and respond in English."
3 PaletteComponent.putMainText("L1", prompt);
4 PaletteLLM.callbackMainText("L1", () => {
5   $("#LLM1-display").show();
6 });
```

The details of the program are as follows:

- Line 1: Retrieves, as a string, the function entered in the input field of the learning material.
- Line 2: Constructs the prompt string and assigns it to the variable `prompt`. For example, if `sin(x)^2` is entered, the resulting string is "`Differentiate sin(x)^2. Format math with $ or $$ and respond in English.`".
- Line 3: Writes the string stored in prompt to the text area of the LLM component.
- Lines 4–6: Sends the prompt entered in the LLM component to the LLM and displays the response after it is returned.

5 Space Optimization for Explanations and Diagrams

5.1 Optimization of LLM Components and Flick Input

In the previous section, we introduced learning material that uses the LLM component to provide explanations of derivatives. However, as shown in Fig. 2 (left), lengthy explanations may not fit within the available display area. Although users can scroll to view the remaining content, learning is more effective when the full derivation is visible at once.

To address this issue, we implemented a mechanism that enlarges the LLM display area by reducing the space occupied by the flick-input interface. The learning material shown in Fig. 2 already includes this function. When the user clicks the "To Portrait" button, the LLM display switches to a vertically extended layout, as shown in Fig. 2 (right). In Standard Mode, the explanation is partially truncated, whereas in Portrait Mode the full explanation fits within the display area.

The JavaScript program executed when the "To Portrait" button is clicked is shown in Code 3.

Code 3: Code for Toggling Layouts

```
1 llm = (-1) * llm + 1;
2 PaletteLLM.setDisplayPos('L1', leftar[llm], topar[llm]);
3 PaletteLLM.setDisplaySize('L1', widthar[llm], heightar[llm]);
4 PaletteButton.setLabel('b1', labelar[llm]);
5 setfmode(fmode[llm]);
```

The details of the program are as follows:

- Line 1: Toggles the value of the variable `llm` between 0 and 1. This serves as the switching mechanism between the two layout states.

- Line 2: Sets the position of the LLM component used to display the response. The arrays `leftar` and `topar`, initialized at startup, store the coordinates for each mode.
- Line 3: Sets the size of the LLM component used to display the response. The arrays `widthar` and `heightar` store the width and height values defined during initialization.
- Line 4: Updates the button label. The array `labelar` contains the strings "To Portrait" and "To Standard," corresponding to the current state.
- Line 5: Switches the flick-input interface between Standard Mode and Compact Mode.

As a preliminary qualitative validation of the proposed system, we presented the learning material to an in-service high school teacher and obtained feedback on its potential classroom usefulness. The teacher noted that flick input is already familiar to many students and emphasized the importance of preserving a record of the learning screen for later review, even in the form of a screenshot. This feedback suggests that fitting the LLM-generated explanation within a single smartphone screen is pedagogically meaningful, because it allows learners to retain the full response easily and revisit it as part of reflective learning. From this perspective, the proposed space-optimization mechanism, which reduces the area allocated to flick input in order to enlarge the explanation display, is important not only for usability but also for supporting review and reflection in mobile learning contexts.

Beyond usability, the LLM-supported explanation function also has important pedagogical implications. Because the system presents not only a final result but also a step-by-step derivation, it may support conceptual understanding by helping learners connect the mathematical expression they entered with the intermediate reasoning process. At the same time, the design of the system provides a partial safeguard against passive reliance on AI-generated solutions, since learners must first formulate and enter the target expression by themselves through the flick-input interface before requesting an explanation. In this sense, the LLM component is intended not as a replacement for the process of learner thinking, but as a scaffold that supports interpretation, review, and reflection. Furthermore, when the full explanation fits within a single smartphone screen, learners can preserve it easily as a screenshot and revisit it later, which may further encourage reflective learning rather than one-time passive consumption.

5.2 Optimization for Interactive Diagrams

Reference [10] describes methods for displaying and interactively manipulating diagrams using CindyScript. However, on mobile devices with limited screen space, diagrams often appear too small for precise interaction, as shown in Fig. 3 (left).

By reducing key size through the resizing function of the flick-input system, the interface can allocate more screen space to visual elements. Figure 3 (right) shows the interface with an enlarged diagram. In this material, the flick-input size is adjusted automatically during page transitions. Consequently, unlike the previous example, no manual toggle button is required, allowing a seamless transition between different stages of a lesson.

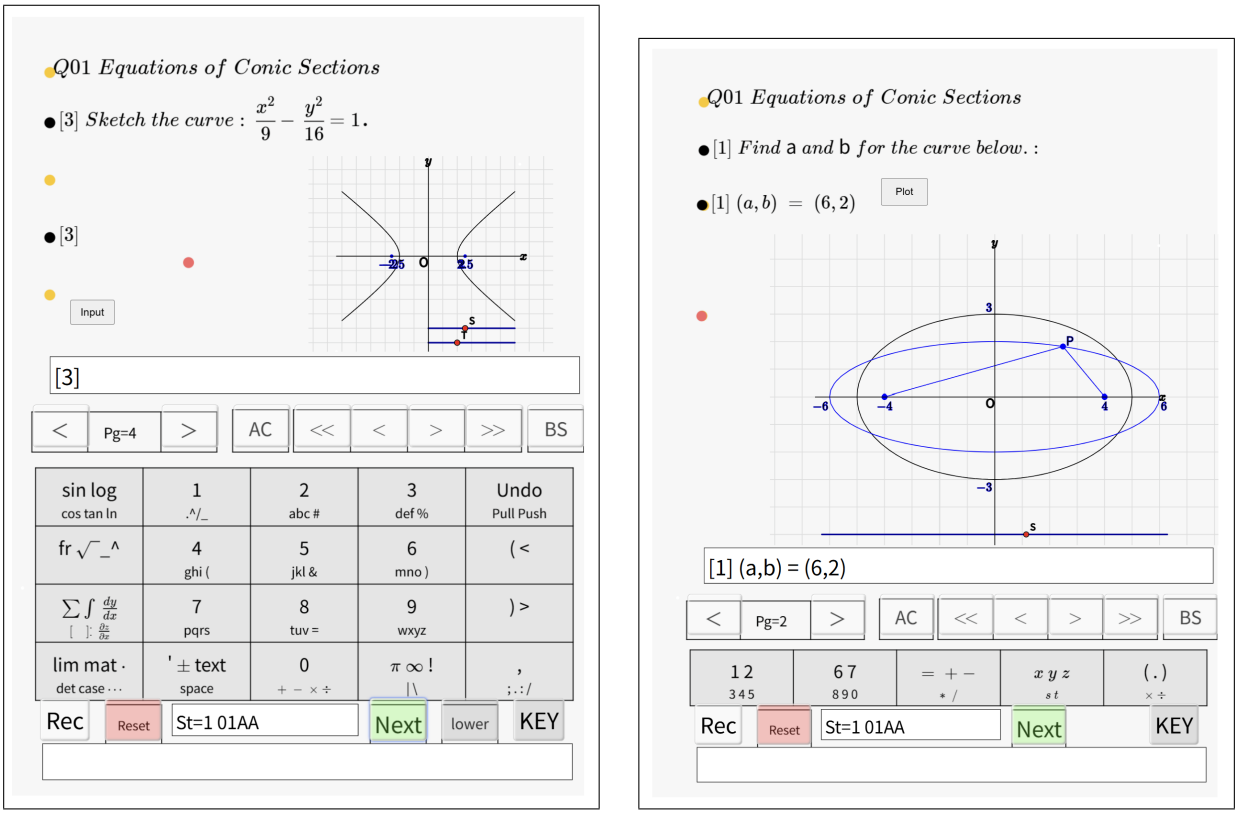

Fig. 3: Interactive diagram in Standard Mode with limited interaction space (left) and enlarged interactive diagram produced through automatic interface scaling (right).

6 Conclusion

This paper has described the evolution of the Flick-Input TeX System from a static input tool into a dynamic, AI-integrated learning platform. By implementing Dynamic Interface Scaling, which supports both manual layout switching through JavaScript and automatic reconfiguration through page transitions, the system addresses the three major barriers to mobile mathematical input: command memorization, input effort, and screen constraints.

The integration of programmable LLM components shifts the system's role from supporting only final answers to supporting mathematical processes through step-by-step derivations, while preserving input usability. In addition, the proposed space-optimization strategies allow interactive diagrams in KeTCindy.JS to remain visible and usable even on small-screen devices. Collectively, these extensions lower barriers to mobile STEM learning and support a robust mobile-native environment that combines intuitive mathematical input with AI-assisted explanation.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Numbers 25K06706 and 21K02752.

References

[1] Nakamura, Y., and Nakahara, T., "Development of a Math Input Interface with Flick Operation for Mobile Devices," in *Proceedings of the 12th International Conference on Mobile Learning,* Portugal, 2016.

[2] Nakamura, Y., "A New Mathematics Input Interface with Flick Operation for Mobile Devices," *MSOR Connections*, vol. 15, no. 2, p. 76-82, 2017. doi:10.21100/msor.v15i2.413.

[3] Nakamura, Y., "Usability Evaluation of Mobile Interfaces for Math Formula Entry," in *Proceedings of the 21st International Conference on e-Society and the 19th International Conference on Mobile Learning*, Portugal, 2023.

[4] Akamine, K., Tsuchida, R., Kato, T., and Tamura, A., "PonDeFlick: A Japanese Text Entry on Smartwatch Commonalizing Flick Operation with Smartphone Interface," in *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, pp. 1–11, 2024.

[5] Kitamoto, T., Usui, H., Hamaguchi, N., Kaneko, M., and Noda, T., "Flick-Input TeX Interface: Lowering Barriers to Mobile Equation Entry," in *Proceedings of the 30th Asian Technology Conference in Mathematics*, Quezon City, Philippines, 2025.

[6] Takato, S., and Makishita, H., "Development of a Question Distribution and Answer Collection System for Mathematics Classes Using Only One Line of Text," in *Proceedings of the 29th Asian Technology Conference in Mathematics*, Yogyakarta, Indonesia, 2024.

[7] Kitamoto, T., Kaneko, M., and Takato, S., "Standalone Web Application for Teachers to Create Teaching Materials on a Browser," in *Proceedings of the 25th Asian Technology Conference in Mathematics*, Bangkok, Thailand, pp. 258–267, 2020.

[8] Kitamoto, T., “Framework for Building an E-Learning System Using Existing Teaching Materials,” in *Proceedings of the 14th MathUI Workshop 2023*, Cambridge, UK, pp. 123–132, 2023.

[9] Kitamoto, T., Kaneko, M., Noda, M., and Kihara, H., “E-Learning Material Creation System That Utilizes Existing Teaching Materials,” in *Proceedings of the 29th Asian Technology Conference in Mathematics*, Yogyakarta, Indonesia, 2024.

[10] Kitamoto, T., Usui, H., Hamaguchi, N., Kaneko, M., and Noda, T., “Exploring the Educational Impact of Cinderella and KeTLMS: Functional Enhancements and Practical Use Cases,” in *Proceedings of the 30th Asian Technology Conference in Mathematics*, Quezon City, Philippines, 2025.

[11] KeTCindy Official Homepage. Available at: https://s-takato.github.io/ketcindyorg/indexe.html (accessed July 19, 2025).