MathAssist: A Handwritten Mathematical Expression Autocomplete Technique

Wenhui Kang
Institute of Software Chinese Academy of Sciences
Beijing, China
School of Computer Science and Technology, University of Chinese Academy of Sciences
Beijing, China
ks_moth@163.com

Jin Huang*
Institute of Software Chinese Academy of Sciences
Beijing, China
School of Computer Science and Technology, University of Chinese Academy of Sciences
Beijing, China
huangjin@iscas.ac.cn

Qingshan Tong
Institute of Software Chinese Academy of Sciences
Beijing, China
School of Computer Science and Technology, University of Chinese Academy of Sciences
Beijing, China
qingshan2017@iscas.ac.cn

Qiang Fu
Institute of Software Chinese Academy of Sciences
Beijing, China
School of Computer Science and Technology, University of Chinese Academy of Sciences
Beijing, China
fuqiang2020@iscas.ac.cn

Feng Tian
Institute of Software Chinese Academy of Sciences
Beijing, China
School of Artificial Intelligence, University of Chinese Academy of Sciences
Beijing, China
tianfeng@iscas.ac.cn

Guozhong Dai
Institute of Software Chinese Academy of Sciences
Beijing, China
dgz@iscas.ac.cn

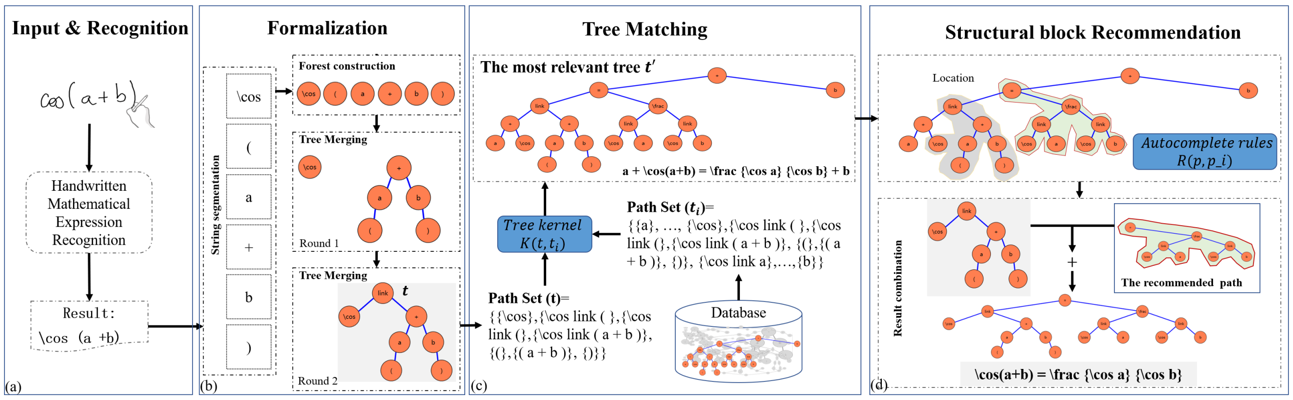


Figure 1: The workflow of MathAssist: a) recognizes user's input strokes and outputs a partial formula in latex; b) builds tree structure according to the latex; c) matches the tree structure in our formula database; and d) selects a structure block (i.e., sub-tree structure) as recommendation from the entire structure of matched formula.

ABSTRACT

Writing and editing mathematical expressions with complicated structures in computer system is difficult and time-consuming. To address this, we proposed MathAssist, a mathematical expression autocomplete technique that recommends full formulas in real-time based on the user's input strokes. Our technique identifies user's input purpose by matching the structure of the current user input to the structure of formulas in a database. To facilitate such process, we propose a novel tree-based formalization to represent formula. In comparison to a mathematical expression recognition algorithm (SRD) and a commercial MicroSoft Ink Equation (InkEqu),

*Corresponding author

This work is licensed under a Creative Commons Attribution-NonCommercial International 4.0 License.

IUI '24, March 18–21, 2024, Greenville, SC, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0508-3/24/03
https://doi.org/10.1145/3640543.3645149

our approach outperformed both of them on task completion time (reduced by 37.14% and 37.58%) and accuracy (32.78% and 10.55% higher). We also discuss our findings in using autocomplete to assist formula editing.

CCS CONCEPTS

• **Human-centered computing → Interactive systems and tools**; **Interaction techniques**.

KEYWORDS

Mathematical expression, Handwritten mathematical expression recognition, Tree-based representation, Autocomplete

ACM Reference Format:
Wenhui Kang, Jin Huang, Qingshan Tong, Qiang Fu, Feng Tian, and Guozhong Dai. 2024. MathAssist: A Handwritten Mathematical Expression Autocomplete Technique. In *29th International Conference on Intelligent User Interfaces (IUI '24), March 18–21, 2024, Greenville, SC, USA*. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3640543.3645149

1 INTRODUCTION

Mathematical expression is an essential tool for formalizing research questions, theories, and solutions in a variety of fields. For a long time, efficiently writing expressions in computing system has been a critical but unsatisfied demand. There are three typical ways of writing and editing mathematical expressions: (1) constructing and editing formulas with icons or widgets; (2) generating formulas with a special Markup Language such as Latex; and (3) recognizing formulas with handwritten mathematical expression recognition. The first two are inefficient and difficult to input special symbols or complex structures contained in mathematical expressions, particularly for beginners [6, 10, 18]. In contrast, recognizing handwritten mathematical expressions is a more natural and effiective way of dealing with mathematical expressions, especially on the mobile or large-screen devices [19, 34, 56].

Handwritten mathematical expression recognition (HMER) techniques have gained increasing concern in the HCI and associated communities recently[47, 66]. To improve the accuracy of HMER, researchers created various methodologies, such as structured recognition methods[3, 8], end-to-end recognition methods[61, 65], etc. Despite this, the most advanced technology can only achieve roughly 60%-80% accuracy on a predefined public dataset with 101 symbols[60, 62], which makes it challenging to meet user requirements[7] and restricts the use of handwritten mathematical expressions in interactive applications. Apart from that, there are other ways to use interactive user interfaces or human involvement to assist mathematical expression recognition. A few of these include online HMER based on human-in-the-loop [22], math boxes for writing mathematical expression [52], and interactive correction for math in OneNote [37]. However, despite the accuracy improved, these techniques generally take longer time to complete the writing.

This paper proposes an alternative approach that incorporates the idea of autocomplete, which is frequently used in text input, into the writing of mathematical expressions, so as to obtain efficient and accurate formula input. In our approach, there are three major challenges in building such an autocomplete technique for mathematical expressions. First, existing formalization method such as Latex are not suitable for matching mathematical expressions. An ideal formalization method for autocomplete technique should not only be capable of presenting mathematical expressions accurately, but also facilitate quick locating and recommending sub-expressions. Second, unlike text input, symbols (i.e., letters, numbers, etc.) in formulas are meaningless and often replaceable. For matching formulas, we should not use methods like edit distance [17] and cosine similarity [27], but rather devise a new way to measure the structural similarity between the formulas. Third, it is difficult to recommend the sub-expression that maintains semantic similarity and coherence with formulas already written from the most relevant mathematical expression is complex. This is because it is not only necessary to match the contextual semantic information of the user input formula with the most relevant mathematical expressions [23, 30], but also to consider the correctness (whether it conforms to the mathematical logic) of the new expressions after autocomplete.

To address these challenges, we proposed MathAssist, a handwritten mathematical expression autocomplete technique using a tree-based formalization to describe formulas during its workflow (Figure 1 (a) and (b)). To determine the purpose of the user's input from the current input strokes, the input strokes are recognized and transformed into tree-based structure form, then matched with existing formulas in a database using an improved tree kernel method [50] (Figure 1 (c)). In order to find an appropriate formula completion that allows users to write formulas quickly and flexibly, we designed recommendation rules that select an optimized structure block from the structure of the matched formula in the last step (Figure 1 (d)). To evaluate our approach, we conducted a within-subject experiment, in which 17 participants completed tasks using MathAssist, a recognition algorithm known as sequential relation decoder (SRD) [62] and a commercial MicroSoft Ink Equation (InkEqu) [12]. Our results showed that MathAssist outperforms SRD and InkEqu in terms of accuracy(32.78% and 10.55% higher) and completion time (reduced by 37.14% and 37.58%). Importantly, we found that accuracy and task completion time remained constant with the increasing formula's lengths in MathAssist. In a semi-structured interview, participants' subjective evaluations are consistent with the quantitative results. The contributions of this paper are as follows:

- We made the first attempt to build a mathematical expression autocomplete technique that recommends the structure blocks of mathematical expression based on input strokes from user.
- We designed a tree-based formalization of mathematical expression, an improved tree kernel method, and a set of recommendation rules to solve the problems of formalizing, matching and recommending mathematical expressions in the autocomplete technique.
- We conducted a within-subject user study to evaluate the performance of the proposed technique, and discussed its rationality in a semi-structured interview.

2 RELATED WORK

2.1 Handwritten Mathematical Expression Recognition

Handwritten Mathematical Expression Recognition (HMER) has been an active research field in pattern recognition since the 1950s, serving as a key component for systems such as physics, geometric theorem proving, and algebraic intelligent tutoring systems [14]. In essence, it functions as a translator, converting handwritten strokes into machine-editable mathematical expressions.

Compared with handwriting text recognition, HMER is a more challenging pattern recognition task, which requires handling similar looks of symbols, special symbols, various 2D nested structures and highly relevant context [14, 55, 64]. Generally, HMER can be divided into three tasks: symbol segmentation, symbol classification, and structural analysis [3, 8, 47]. Traditional multi-stages approaches solve these three problems with either sequential or global methods. For the sequential methods [2], the activities of each step are carried out successively in sequential ways, with the output from one stage serving as the input for the next. Errors will, of course, be propagated in the same way. For global methods [3, 8], these tasks are handled simultaneously with global context information. As a result, this also makes global methods more complex. Additionally, both kinds of methods require domain knowledge.

To address the issues with traditional multi-stage methods, researchers combine the three sub-problems of HMER into one problem (called stokes-to-sequence or image-to-sequence problem). End-to-end methods with encoder, decoder and attention module as key components were used to solve this problem [28, 63]. For example, "watch, participate, and parse (WAP)" method [63] used a convolutional neural network to encode input handwritten traces, and a recursive neural network decoder to generate mathematical expressions. "Sequential relation decoder (SRD)" model [62] employed a RNN encoder with gated recurrent units to model the input strokes, and a sequential relation decoder with attention model to generate the LATEX expressions. "Track, attend and parse (TAP)" method [61] used a stack of bidirectional recurrent neural networks with gated recurrent units to recognize the handwritten mathematical expressions. Position correction attention mechanism [16], drop attention module [28], coverage model [49], self-attention [11], and multi-head attention [14] are used to address the issues of imprecise attention distribution in the decoder and different sizes of symbols. To enhance the localization and classification of the high-level features in HMER, a symbol classifier and beam search process [54] are employed in the encoder-decoder framework. Additionally, tree-based decomposition and sub-expression interchange [53], temporal alignment of the input feature sequence and corresponding symbol label sequence [40], pattern generation strategy [24], bi-directional mutual learning [9] and relation-based sequence representation [41] are used to improve recognition capability with data augmentation.

Overall, the embodiment of recognition capability in HMER has undergone seismic shifts as technology and software based on artificial intelligence continue to advance. Affected by some limitations, which contains their limits in varied strokes styles[46] and stroke order variations[26], their inability to accurately recognize indistinguishably similar symbols [49], horizontal and subscript problems[4], and other concerns [13], the average expression accuracy on the public available datasets (CROHME 2014, CROHME 2016, and CROHME 2019) can only reach about 60%-80%. In our technique, we minimize these problems through autocomplete technique.

2.2 Autocomplete technique

Autocomplete is a technique that presents a list of suggestions to the user based on their input in a specific order [29, 39]. Autocomplete has been widely used in text editing, coding, and information search to enhance the user experience by accelerating text entry and reducing spelling errors [5, 15, 21, 44]. In these applications, autocomplete is mainly targeted at flat text or string data. With the development of artificial intelligence and big data, autocomplete gradually expands to prototype virtual breadboard circuits [29], painting repetition [59], auditing [44], animated sculpting [42], aggregate elements [20]. But these autocomplete are mainly used for tasks that already have a fixed workflow, like the workflow in 3D sculpting often consist of predictable brush choices and operations [43]. It needs to be said that both of them rely on the determination of how similar the user input and the autocomplete results, such as graph similarity is used to measure the elements consisting of multiple samples [20], composite similarity consisting of samples, operations and neighborhood to autocomplete repeated paintings [59], edit distance and cosine similarity [17, 27] is common in text-based autocomplete.

It is intuitive to use autocomplete to accelerate the input of handwritten mathematical expressions. However, we found no previous attempts to develop and evaluate such systems. This could be related to the fact that there is a lot of room for improvement in current HMER accuracy, which has led to the majority of research focusing on developing new algorithms. There are a few researchers who introduce human-in-the-loop [22] or intelligence widgets [52] to improve mathematical expression recognition, but these methods generally increase the completion time of writing formulas. Overall, current HMER research is focused on developing new algorithms for better recognition, with little emphasis on applying them to mathematical expression autocomplete.

2.3 Formalization of mathematical expression

The formalization of mathematical expressions is mainly classified into string-based [1, 38] and tree-based [31, 45]. The string-based formalization is to translate mathematical expressions into flat strings in accordance with specific rules, which are often used to write mathematical expressions in code [1], or to parse mathematical expressions in HMER [55]. However, it is difficult to express the structure of mathematical expressions and their relationships between symbols using the string-based formalization. The tree-based formalization, as intermediate data for mathematical expression processing, can effectively express the structures of mathematical expression, such as in [64], they constructed a symbol relation tree based on six spatial relationships ("Above", "Below", "Right", and others), where the nodes in this tree are the input strokes and the connections between the nodes represent the spatial relationships between input strokes. In [3], a parse tree is given to describe the relationship between input sequences of strokes and mathematical expressions, where its leaf nodes are input strokes that make up the single terminal symbol (like "x", "+"), other non-leaf nodes are non-terminal symbols (like "Sym", "Exp"), and the connection of nodes represents the derivation process of non-terminal symbols to terminal symbols. The previous tree-based formalization can not emphasize the structures of the mathematical expressions themselves and cannot distinguish the attribute of the symbols in mathematical expressions, where the symbols' attribute contains operand, operator and delimiter. In this paper, we design a new tree-based formalization to autocomplete mathematical expressions.

3 TREE-BASED FORMALIZATION OF MATHEMATICAL EXPRESSION

We design a method to convert the formalization of mathematical expression from mathematical markup language (Latex) to a tree-based structure (called inorder tree), which uses the trees/sub-trees to represent relationships among different structures for quickly locating and recommending structures and sub-expression in mathematical expression. Different from symbol relation tree [3, 31, 62, 64] constructed based on strokes' or symbols' spatial relationships, such as Above, Right, etc., the inorder tree [48] emphasizes structural blocks of mathematical expression. To achieve this, we use the delimiter, a pair of symbols that separate text strings such as braces ('{' and '}'), in the Latex string to determine delimitation structures in the mathematical expression, then construct the inorder tree with inorder traversal. The delimitation structure mentioned above is a structure made up of symbols or sub-expressions that are enclosed by a pair of delimiters, which is used to distinguish structures between current and previous sub-expressions. In addition, inorder traversal can ensure the bottom-up search information conforms to user's manner of writing and parsing mathematical expressions from left to right, enable that the traversal result is consistent with Latex string, and enhance the ability to predict the semantically related symbols and structures of user's subsequent input.

In details, we build the inorder tree as follows:

- `String segmentation`: Segment Latex string to separated symbol from left to right with forward maximum matching method [51] in 101 math symbols.
- `Making tree node`: Convert each symbol into a node and mark the node with level 0. The order of these nodes is the same as symbols in the Latex string.
- `Obtaining delimitation structures`: Traverses all the nodes from left to right, finds all pairs of delimiters, defines the nodes enclosed by a pair of delimiters as a delimitation structure.
- `Set the level for nodes in delimitation structures`: Increases the levels of all nodes in a delimitation structure by 1. If there are nested delimitation structures inside a delimitation structure, nodes in the nested delimitation structures should be further increased by 1.
- `Tree merging`: Traverse all the nodes, starting from the one with the highest level, merge adjacent binary nodes that have the same levels into one tree, and decrease the level of the merged tree by 1. The merged tree is then treated as a node in the next repeat.
- `Repeat`: Repeat the "Tree merging" step until there is only one node left with level equal to 0.

The merging in "tree merging" can be described as: given the adjacent merged nodes $\{t_i, ..., t_j\}$ with same level, adding $\{t_{i+1}, ..., t_j\}$ to parent node (referred to as a root) of t_i or root's right child in turn. If root's right child exists, it will be added to root's parent node. And repeat it until the subsequent $j - i$ trees have merged from bottom to top. In this process of sequential merging, we set some tips to better construct a inorder tree. Firstly, we introduce a "link" node between two merged nodes for connection if they contain symbols with the same attribute or if the node inserted at the root is a merged tree. Second, if there are paired delimiters, we first merge symbols between delimiters and then insert delimiter into child nodes of the leftmost and rightmost leaf nodes of the merged tree. Third, if a fractional structure exists during the traversal of nodes in the tree merging, such as "\frac {...} {...}", we merge them in a "numerator-fraction-denominator" order rather than the Latex-based order, making the numerator and denominator become left and right subtrees of the fractional symbol. Finally, the merged tree is then treated as a node in the next repeat, and its symbol's attribute is the attribute of the symbol corresponding to its root node. The tree-based formalization of $1 - \cos(a + b) = \frac{1}{3}$ is shown in Fig 2.

In the follow, some terms utilized in our approach are defined:

- `Inorder Tree`: it is made up of numerous subpath with symbol-based nodes. Its formal description is,
$$t = \{\boldsymbol{V}, \boldsymbol{P}\}$$
where $\boldsymbol{V}$ is nodes of tree, and $\boldsymbol{P}$ is the all subpath of tree.
- `Subpath`: A subpath is a sequence of inorder traversals between two leaf nodes in the inorder tree. Starting with the leftmost leaf, subpaths from each leaf node to other nodes form the subpath set, as shown in Figure 3. The subpath set of the inorder tree can be expressed as
$$\boldsymbol{P} = \{p_0, ..., p_n\}$$
where $\boldsymbol{P}$ is a subpath set that includes all of the subpaths. n is the number of subpaths.
- `Trunck`: the trunk of the tree is the path from the root node to the leftmost leaf node, and it is directly connected by each left child in turn, As shown in Figure 3.
- `The most relevant tree`: it is a inorder tree constructed by a mathematical expression in the database, which can best match the currently written mathematical expression in terms of structures and symbols.
- `Structural block`: A structural block is equivalent to a sub-expression contained in mathematical expression, which consists of two operands and an operator connecting these two operands. If only one operand exists, it can also be seen as a structural block.

For the inorder tree, its characteristics are that (1) a node in inorder tree additionally provides details about strokes, priority and symbol attributes that belong to this symbol. (2) The delimiter nodes are kept when formalizing the tree to ensure consistency between tree and Latex formula exchange, especially for complicated structures with several nested delimiters. (3) Separation of operand operators, all leaf nodes of the inorder tree are operand

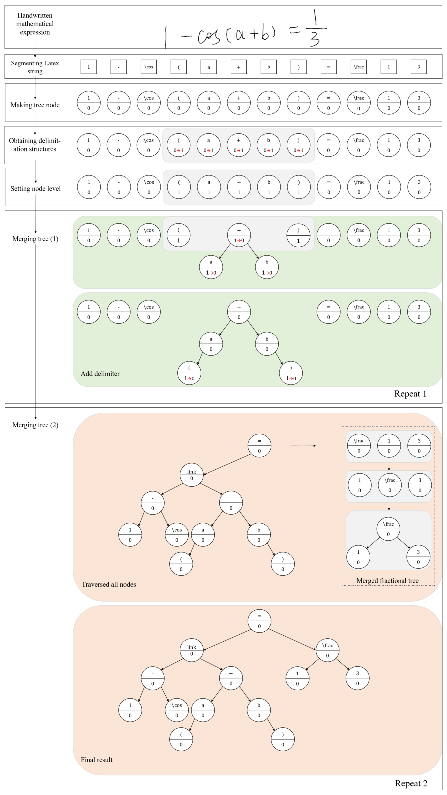
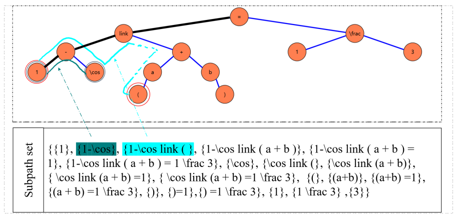


Figure 2: The tree-based formalization about " $1 - \cos(a + b) = \frac{1}{3}$". A circle represents tree's node, wherein a symbol on horizontal line is single Latex symbol, and a number under line denote its level.



{{1}, {1-\cos}, {1-\cos link (}, {1-\cos link (a + b)}, {1-\cos link (a + b) = 1}, {1-\cos link (a + b) = 1 \frac 3}, {\cos}, {\cos link (}, {\cos link (a + b)}, { \cos link (a + b) =1}, { \cos link (a + b) =1 \frac 3}, {(}, {(a+b)}, {(a+b) =1}, {(a + b) =1 \frac 3}, {)}, {)=1},{) =1 \frac 3}, {1}, {1 \frac 3} ,{3}}

Figure 3: The subpath set of "$1 - \cos(a + b) = \frac{1}{3}$", in which one subpath is equivalent to a sequence of inorder traversals between two leaves. The black paths in the tree are called trunks.

nodes and all non-leaf nodes are operator or "link" nodes (if the delimiter exists, its node will appear on the leftmost and rightmost leaf nodes of the inorder tree merged based on the nodes in the delimitation structures). By doing so, the formula's primary structure will be reflected in the tree's trunk, allowing for easy location and recommendation; (4) Except for structure block containing fractional symbols, the result of the inorder traversal of the corresponding symbols in a inorder tree is a Latex string of mathematical expression. (5) The right subtree of all nodes on the trunk is a single symbol or structure blocks based on delimitation structure. Finally, child nodes of two trunk nodes with the same priority may be switched to one another.

4 MATHASSIST

4.1 The framework of MathAssist

The framework of MathAssist is shown in Figure 4. It mainly consists of: (1) an input module for users to write required handwritten strokes; (2) a recognition module, which provides recognition results of current strokes with an end-to-end method; (3) a autocomplete module, which is mainly for assisting users in effectively writing their required subsequent expressions through solving the tree matching and structural block recommendation problems mentioned in the previous section; and (4) interaction module that allows user to accept, reject and edit autocomplete results provided by the autocomplete module. The four modules work together in such a way that once user writes mathematical expressions in input modules, the recognition module receives and recognizes them, and then sends their recognition result to autocomplete module. After obtaining the recognition result, the autocomplete module finishes its task and then provides the corresponding autocomplete result to user. Finally, user can process the autocomplete result through the interaction module and interact with it in accordance with his actual demands or can write new strokes into the input module.

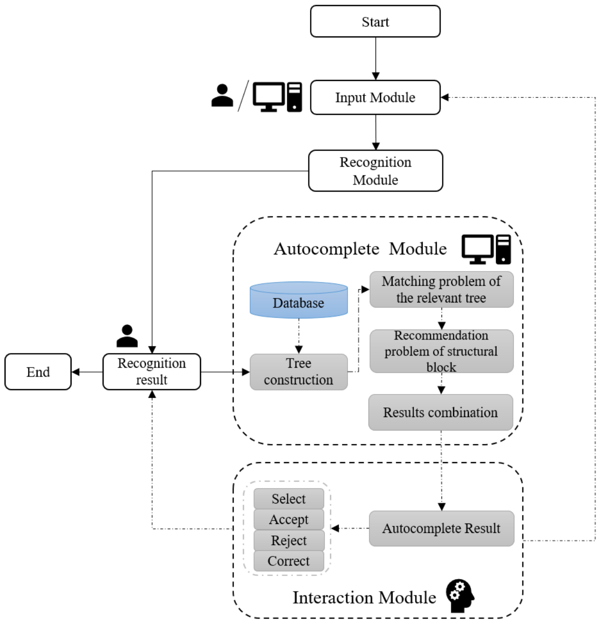


Figure 4: The framework of MathAssist, in which interaction module presents only the interaction with autocomplete result

In addition, the system keeps going through the entire procedure for the user until there is no more input or interaction. In the following subsections, we go into detail about each of the four modules.

4.2 Input Module

We employ the input module to receive both user's handwritten strokes and autocomplete strokes, wherein the autocomplete strokes are the handwritten strokes that MathAssist anticipates the user will write later. To make it easier to distinguish between them before accepting the autocomplete strokes, they are provided in light gray, whereas the user's handwritten strokes are presented in black. Only after accepting or modifying the autocomplete strokes will it turn black, otherwise it will be removed from the input module.

4.3 Recognition Module

The purpose of the recognition module is to recognize handwritten strokes from input module and transmit recognition result to the autocomplete module. It will work once the user writes new strokes or modifies already-existing strokes in the input module.

In this module, we used the "sequential relation decoder (SRD)" model [1], which outperforms other online handwritten mathematical expression recognition with better accuracy [62]. SRD is built on an encoder-decoder framework, where it employs a RNN encoder with gated recurrent units to model the input strokes, and a sequential relation decoder with attention model to generate the LATEX expressions. In essence, the SRD model can be viewed as a converter from strokes to string-based expressions, and the accuracy of this conversion gradually declines with the increase of the number of strokes [62].

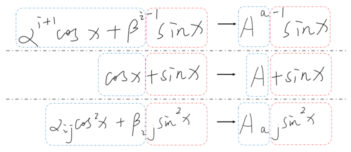
To decrease the recognition errors caused by the increase of strokes, we reorganize new input and standard strokes before the subsequent round of recognition, in which standard strokes take over the strokes of mathematical expressions that have been correctly recognized or automatically completed, as shown in Figure 5. There are three different types of standard strokes (the corresponding mathematical expressions are A^a, A and A_a, the three are picked as they are easy to recognize and do not introduce other errors) in our approach that are designed to match bottom number in the preceding round of recognition results and reconstruct entire results, especially for multi-nested super/subscript structure. The matching and reconstruction process is equivalent to substitute A or a in outcome corresponding to the standard strokes. As for how to select standard strokes, it depends on which of the three is consistent in structure with the last structural block of the correct mathematical expression.

4.4 Autocomplete Module

Based on mathematical expression database and recognition result for current input strokes, this module introduce autocomplete technique to address two problems: matching problem of the most relevant tree and recommendation problem of structural block, allowing user to write mathematical expression more quickly and accurately.

4.4.1 Database. Our database is part of CROHME 2016 competitive dataset, which is the most widely used for online HMER [32, 62, 66]. The dataset has 9983 handwritten mathematical expressions and

[1]https://github.com/JianshuZhang/SRD



$\alpha^{i+1}\cos x+\beta^{i-1}\sin x$ — $A^{a-1}\sin x$
$\cos x+\sin x$ — $A+\sin x$
$\alpha_{ij}\cos^2 x+\beta_{ij}\sin^2 x$ — $A_{a_j}\sin^2 x$

Figure 5: Strokes reorganization: standard strokes replace strokes of mathematical expressions that had been correctly recognized or automatically completed (in blue box), where strokes in the red dotted line box represents the newly input strokes.

101 math symbol classes. It is distributed as a set of Ink Markup Language (InkML) files, each of which contains: ground-truth in LATEX and MathML formats; strokes; the segmentation and assigned labels of each symbol in the expression. In our experiment, mathematical expression with strong semantic associations, contained summation expressions, integral expressions, limit expressions, logarithmic expressions and trigonometric functions, were selected as data source to obtain more reasonable autocomplete results, which together account for around 36% of the all dataset. What is novel is that, based on the fundamental logic and common sense of mathematical expressions, we divide 101 symbols into operators, operands, and delimiters to indicate symbols' attribute, and set priority weight among them.

4.4.2 Matching the most relevant tree. In our approach, the matching problem of the most relevant mathematical expression can be described as that: given a inorder tree t constructed based on the current input strokes and its corresponding string-based recognition result, find the most relevant tree t' among all the inorder trees T according to the function $K(t, t_i)$.

$$t' = \arg\max_{t_i \in T}(t_i | K(t, t_i))$$

where t_i is the i^{th} inorder tree in knowledge base.

To address this matching problem, we design a tree kernels for measuring the relevance between two inorder trees. The idea behind this kernels is to convert a inorder tree into subpaths in our designed way that enables the kernel to operate on the tree directly. For example, the subpath set of $1 - \cos(a + b) = \frac{1}{3}$ is shown in Fig 3.

Let us assume that we define a tree kernel $K(t, t_i)$ between t and t_i with the subpath sets, it can be defined as:

$$K(t, t_i) = \sum_{p \in \boldsymbol{P'}} num_p(t) num_p(t_i) w_{|p|}, t_i \subseteq \boldsymbol{T}, \boldsymbol{P'} = \boldsymbol{P_t} \cup \boldsymbol{P_{t_i}} - \boldsymbol{P_t} \cap \boldsymbol{P_{t_i}}$$

where $num_p(t)$ is the number of times subpath p occurs in tree t, $\boldsymbol{P_t}$ is the subpath set of tree t, $\boldsymbol{P'}$ is the set of all subpaths set in t and t_i. The weight w_p of a subpath p is measured as:

$$w_p = w_{p_t} \cdot w_{p_{t_i}}$$

where w_{p_t} and $w_{p_{t_i}}$ is the weight of subpath in tree t and t'. To make the weight larger as the frequency of subpath occurrence increases, we decide the weight of tree t via:

$$w_{p_t} = \begin{cases} -log_2(num_p(t)) + 1, num_p(t) \geq 1 \\ 0, num_p(t) = 0 \end{cases}$$

To reduce time complexity of tree kernels, we employ auxiliary stacks and prefix arrays, which are used to store inorder traversal nodes and subpaths, as well as a multikey quicksort algorithm to achieve linearthmic time $O(|V_t|log|V_{t'}|)$ on average, where $|V_t|$ is the number of nodes of tree t.

4.4.3 Recommending the structural block. The structural block recommendation can be described that is to select the structural block's subpath p' in the most relevant tree t' that best matches the path p of inorder tree t according to the designed autocomplete rules $R(p, p_i)$.

$$p' = p_i, s.t. R(p, p_i), p \subseteq \boldsymbol{P_t}, p_i \subseteq \boldsymbol{P_{t'}}$$

where p is a path of t, p_i is the i^{th} subpath in t', $\boldsymbol{P_t}$ and $\boldsymbol{P_{t'}}$ are subpath sets of t and t'. Final, the structural block's path $\hat{p}$ is

$$\hat{p} = p' - p$$

.

In details, the rules R between t and the most relevant tree t' is determined by location L, continuous value C and priority S.

$$R(p, p_i) = \{\boldsymbol{L}, \boldsymbol{C}, \boldsymbol{S}\}$$

where $\boldsymbol{L}, \boldsymbol{C}, \boldsymbol{S}$ is calculated as follows:

- L: Locate the position $\boldsymbol{L} = \{v_i, ..., v_j\}$ of t in t' based on a equivalent replacement.
- C: Check the node corresponding to L is continuous in inorder traversal. If not, a subpath from v_i to v_j in t' is p', and the path consisting of the remaining nodes after removing $\boldsymbol{L}$ from p' is final recommended result $\hat{p}$.
- S: Search best subpath in t' if the position is continuous. Based on $\boldsymbol{L}$, we compare the priority between symbols represented by the previous node of v_i and the next node of v_j, and then extend a structural block outward from the node (v_i or v_j) with high priority. Finally, p' is a subpath combined nodes in L and extended structural block, $\hat{p}$ is subpath composed of the extended structural block and the node with high priority.

About the equivalent replacement of the inorder tree t in the most relevant tree t', as shown in Figure 6, we determine it only in three cases: (1) t' contains subpath p_r which is a subpath holding the rightmost leaf node in t; (2) t' does not contain subpath p_r, but it contains the same subpath as $p_s = \{p_i | p_i \subseteq \boldsymbol{P_t}, p_i \neq p_r\}$ of tree t, and nodes that follow it have the same structural attribute; (3) there is a subpath in t' that have same structural attribute as the last structural block of t. Where the difference of operators make significant distinct of expressions' structure. In our recommendation, we use the priority sequence of symbols within subpath to express the structural attribute.

4.4.4 Results Combination. Based on the relationship among p, p' and $\hat{p}$, we provide users with 5 candidate results. One of them is the preferred result, containing a combined Latex string and corresponding autocomplete strokes, and the other four alternative results provide strokes only if they are selected. Because the output of Latex string is an overall replacement of old string, the it is

$$\begin{array}{ll} \boxed{\sin^2\theta} & \\ \boxed{\sin^2\theta} + \cos^2\theta & (1) \end{array}$$

$$\begin{array}{ll} \boxed{\sin^2\theta}\boxed{-\sin^2\theta} & \\ \boxed{\sin^2\theta}\boxed{-\cos^2\theta} = cos2\theta & (2) \end{array}$$

$$\begin{array}{ll} cos\boxed{\alpha - k} & \\ cos\boxed{x\ +i}\sin x & (3) \end{array}$$

Figure 6: Some cases of equivalent replacement, where it has same subpath in red boxes and same structural attribute in blue boxes. The most relevant tree constructed by recommended result (the second formula in each row) contained: (1) the subpath holding the rightmost leaf node from the inorder tree constructed by recognition result (the first formula in each row), (2) the same subpath and nodes that follow it have the same structural attribute, (3) the same structural attribute as the last structural block from the inorder tree constructed by recognition result.

equivalent to inorder traversal result of symbols in path p'. Stroke information differs from Latex in that it outputs the autocomplete strokes while preserving the original input, which is equivalent to the presentation of the strokes corresponding to the inorder traversal symbols in path $\hat{p}$. To keep their size and placement consistent with the current input, the autocomplete strokes must be scaled and moved.

4.5 Interaction module

This module provides users with some necessary ways to interact with MathAssist, detailed manifestation of which is direct interaction between users and their input strokes or autocomplete results. Throughout the interaction process, users can have multiple rounds of interaction in two ways. One is users' modification of their own written strokes, including undo, redo, erase and clear. The other is the interaction with autocomplete results, which mainly contains acceptance, rejection, selection and correction, where acceptance means that user thinks the autocomplete result is completely correct, rejection means that user thinks it is incorrect or takes more time to modify than to continue writing, selection allows user to select the result they want from the candidate results, and correction indicates that user thinks it is similar to what he wants and that it is easier to correct than to continue writing.

The four interaction are that, acceptance is primarily initiated by clicking on the autocomplete result or "Accept" button, rejection is caused by continuing to write or by clicking "Reject" button, selection is done by touching down/up, while correction is consistent with interaction of one's writing strokes.

5 USER STUDY

To evaluate MathAssist, we compared it with two baseline methods: the sequential relation decoder (SRD) and the MicroSoft Ink Equation (InkEqu). As mentioned earlier, the SRD is an end-to-end state-of-the-art HMER algorithm, and it also contributes to the recognition module in our approach. By comparing MathAssist with SRD, we can access whether the introducing autocomplete technology can improve the existing HMER algorithm. The other baseline, InkEqu, is a commercial HMER that can correct misrecognized ink. We compared our approach with the InkEqu to further discuss the feasibility of using autocomplete technique in formula editing. Twenty mathematical expressions divided by two length levels (i.e., short and long) were selected as the test bed, and the three input methods were compared in terms of accuracy, completion time, and subjective rating. The study was designed to answer the following research questions:

- RQ1: How does MathAssist differ from the other two methods in terms of accuracy and completion time?
- RQ2: What effect does the length of equation have on the performances of the three methods?
- RQ3: How do users feel about MathAssist in terms of accuracy, efficiency, helpfulness, satisfaction, preference, and cognitive load?

5.1 Apparatus

The evaluation of our experiment was conducted on a ThinkPad X1 Yoga gen6 with an 11th Gen Intel® Core™ i7-1165G7 CPU (2.80GHz) having 16.0G of RAM and the 14-inch, 1920×1200 Pixel Sense display, running Microsoft Windows 10 Professional. The device was equipped a pen and a reversible touch screen which support with 10 touch points. It was rested upon a table and pitched at an adjustable incline so that participants could sit and work comfortably. The recognition method is run on the device which has an Intel® Xeon® Silver 4114 CPU (2.20GHz) having 16.0G of RAM, 2 × GeForce RTX 2080 and Ubuntu 18.04 system.

5.2 Participants

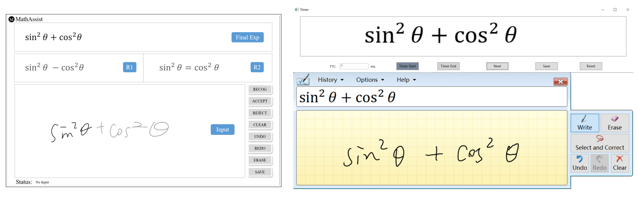
We recruited 17 participants (6 undergraduate students and 11 graduated students, and all of right hand) ranging in age from 21-39 (M=26.47, SD=4.29). 9 participants are male and 8 are female, and only one of the males had no experience with digital pens and handwriting devices. The participants' backgrounds included designer, teacher, IT worker and researcher.

5.3 User Interface

To better compare the performance of MathAssist with the baseline methods for handwritten mathematical expression input, we built a user interface illustrated in Figure 7(a). Our user interface consists of four widgets: display widget (Up), stroke input widget (Lower left, Input), interactive widget (Lower right), and status widget (Bottom). The Stroke input widget is mainly used for user to write handwritten mathematical expressions or accept autocomplete mathematical expression strokes. The display widget shows user some standard typesetting mathematical expression, which contained a display area of final outcome (called Final Exp) and two display areas of other alternative recommended results (called R1 and R2) in the candidate results. If the desired mathematical expression appears in R1 or R2, the user can click to select it. As for the status widget, it shows the system's current state, such as RECOGNITION, INK ENTRY, SELECT, and so on.

Eight different ways to interact with input keys or autocomplete results are available in the interactive widget, and each one is activated by clicking a button. The commands "RECOG" and "ACCEPT" and "REJECT" allow the user to accept or reject an autocomplete result, respectively, while "SAVE" is used to save the strokes in the input widget and the Latex string in "Final Exp" into a file. The remaining four interactions are mostly for the input widget's strokes. "UNDO" and "REDO" are used to reverse or undo the most recent stroke, while "ERASE" and "CLEAR" are used to remove individual strokes from the input widget or to clear all strokes.

We used the same user interface in a comparison method SRD. The only difference is that there are no results in "R1" and "R2", and "Final Exp" merely displays a recognition result based on user's input strokes, while no strokes about subsequent strokes appear in the input widget. For InkEqu, we used the default user interface as shown in Figure 7(b). Participants were free to use all of InkEqu's features, including "Undo", "Redo" and "Select and Correct" for accelerating the edit. To monitor task completion time with InkEqu, we designed a timing tool with three main functions, including "Timer Start","Timer End", and "next".



(a) (b)

Figure 7: The user interface for the three techniques. (a) The user interface for SRD and MathAssist, contained display widget (Final Exp, R1 and R2), stroke input widget (Lower left, Input), interactive widget (Lower right), and status widget (bottom). (b) The user interface for InkEqu, its source: https://www.microsoft.com/en-us/microsoft-365

5.4 Procedure

The procedure for the experiment was as follows: First, participants familiarized themselves with the three methods; second, the participants used the three techniques to complete a mathematical expressions writing task (detailed in the next section); third, participants tried write any other mathematical expressions freely in an open task; finally, the participants fill up a post-questionnaire and follow with a semi-structure interview. The post-questionnaire, as shown in Table 1, asked for participants' feeling about accuracy, efficiency, helpfulness, satisfaction, preference, and cognitive load on the three techniques using a 5-point scale [25, 36, 52, 65]. Also, we asked participants some question in post-questionnaire about whether it was reasonable to add autocomplete in the expressions writing, whether MathAssist introduced some troublesome such as interrupting their thinking or difficult interaction, and could MathAssist help them to understand their required expressions. Finally, we ask the participants about their experience and attitude towards MathAssist in the interviews. Example questions are like:

- What do you think of the MathAssist?
- How do you think the autocomplete capabilities when using MathAssist to write mathematical expressions?

Table 1: The post-questionnaire for the three techniques

ID	Do you agree with the provided conclusions? 1 = strongly disagree, 5 = strongly agree
Q1	The technique can accurately recognize handwritten mathematical expressions.
Q2	The technique is efficient in recognizing handwritten mathematical expressions.
Q3	I am satisfied with the technique's performance.
Q4	I prefer this technique out of the three.
Q5	The technique is helpful.
Q6	I feel that using the technique requires relatively higher cognitive load.

- What machine errors or limitations do you find unacceptable?
- Not limited to MathAssist, any other suggestions for autocomplete technique?

Each participant took about 70 minutes to finish the experiment and were compensated ¥150 for their time.

5.5 Mathematical Expressions Writing Task

In the mathematical expressions writing task, a participant was asked to write 20 mathematical expressions three times, once for each method. Those mathematical expressions are shown in Table 2 ordered by the number of symbols (symbols like tan, sin, etc. are considered as one symbol), which contains summation expressions, integral expressions, limit expressions, logarithmic expressions, and trigonometric functions. The length of mathematical expressions is divided into short and long depending on the number of symbols and strokes. The short mathematical expressions have less than 12 symbols and fewer than 20 strokes, whereas long mathematical expressions have more than 12 symbols and more than 20 strokes. Each length has ten different mathematical expressions. For each participant, the order of which method was employed first was alternated. During the task, participants could correct clerical errors or potential errors through the provided interactive tools to attain more accurate results. If participants were still unable to obtain their desired result after modifying the recognition or autocomplete result multiple times, they could end the current task and move to the next task. There is no explicit time limit on the input of one mathematical expression.

5.6 Collected Data and Metrics

We recorded the recognition result, written time, recognition time, autocomplete time and interactive time for each expression. We also recorded all of the inputed strokes for each expression.

Expression Recognition Rate (ExpRate) [66] and Time To Completion (TTC) [52] were computed to evaluate the performances our method in accuracy and time.

ExpRate is defined as the percentage of correctly recognized mathematical expressions matching ground-truth up to the symbols, relations, and structure:

Table 2: Mathematical expressions used in the evaluation

Num	Mathematical expression
E1	$n\log(n)$
E2	$\log_2 8 = 3$
E3	$\tan z = \frac{\sin z}{\cos z}$
E4	$\cos\theta_1 + i\sin\theta_1$
E5	$\int_a^b \frac{\sqrt{x}}{2}dx$
E6	$\sin^2\theta + \cos^2\theta = 1$
E7	$\cos x + i\sin x = e^{ix}$
E8	$e = \sum_{k=0}^{\infty}\frac{1}{k!}$
E9	$\sum_{i=2n+3m}^{10} ix$
E10	$\int_{\log 3}^{0}\frac{1}{e^t+1}dt$
E11	$\log_2 8 + \log_3 9 + \log_4 16$
E12	$\frac{\tan\alpha - \tan\beta}{1+\tan\alpha\tan\beta}$
E13	$\pi\int_c^d \{g(y)\}^2 dy$
E14	$\lim_{n\to\infty} t_{3n} = \frac{1}{2}\log 2$
E15	$\lim_{x\to\frac{\pi}{2}+0}\tan x = -\infty$
E16	$\log z = \log r + i(\theta + 2n\pi)$
E17	$\cos(a+b) = \cos a\cos b - \sin a\sin b$
E18	$\lim_{x\to\infty}\int_0^x e^{-y^2}dy = \frac{\sqrt{\pi}}{2}$
E19	$a(n) = \sum_{k=1}^{n}(-1)^{n-k}k!$
E20	$\lim_{n\to\infty}\frac{1}{n}\sum f(\frac{r}{n}) = \int_0^1 f(x)dx$

$$ExpRate = \frac{Number\ of\ correctly\ recognized\ expressions}{Total\ number\ of\ expressions}$$

The ExpRate is a recognized metric for evaluating the performance of handwritten mathematical expression recognition used in the CROHME competitions [33, 58, 61]. Instead of characterizing partially correct recognition in words or structures, the ExpRate can enables us to concentrate on assessing whether the final result is precisely correct.

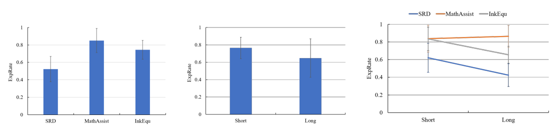
TTC is the total time spent from the beginning of the input formula to its ending.

6 DATA ANALYSIS

To perform statistical analysis, we organized the collected data into a dataset consisting of two independent variables: Technique (SRD, MathAssist and InkEqu) and Length (short and long), and two dependent variables: ExpRate and TTC. Specifically, ExpRate and **average** TTC for one participant in one Technique × Length condition (10 mathematical expressions) were treated as samples in the dataset. As a result, the dataset consisted of 3 Technique × 2 Length × 17 Participant values. The analyses of the two dependent variables were performed separately according to the same procedure as follows:

First of all, a Kolmogorov-Smirnov test (alpha = 0.05) was used for normality test. Then, repeated measures ANOVAs and post-hoc multiple comparisons with Bonferroni were performed if the data were normally distributed. For the data that were not normally distributed, an ART method was used to data transformation before statistical analysis as suggested in [35, 57]. In addition to investigating the effects of Technique and Length on ExpRate and TTC, we also performed paired comparisons of the five subjective indicator among the three techniques. In this case, paired t-tests were used for normally distributed data.

6.1 Expression Recognition Rate



(a) ExpRate under three techniques (b) ExpRate at two length levels (c) ExpRate of the three techniques at two length levels

Figure 8: Comparison of all formulas' ExpRate.

The result of ExpRate is presented in Figure 8 Repeated-measure ANOVA showed a significant main effect of Technique (F(2,14)=35.724, p<0.001, partial η^2=0.691), but not on Length (F(1,15)=1.584, p=0.226, partial η^2=0.090). In paired comparison on Length, it was found that there was significant difference in MathAssist and InkEqu (p<0.01), but not on SRD (p=0.868). Interaction effect between Technique and Length was significant (F(2,14)=17.659, p<0.001, partial η^2=0.525).

As shown in Figure 8(a), MathAssist achieved the highest ExpRate of 85.05% (SD=0.14), followed by 74.5% (SD=0.11) for InkEqu and 52.26% (SD=0.15) for SRD. Post-hoc tests showed that the differences between MathAssist and SRD, between SRD and InkEqu are all significant (all p<0.001), but there is not significant differences between MathAssist and InkEqu. As shown in Figure 8(b), participants generally had lower input accuracy in long formulas (64.77% (SD=0.14)) compared to short formulas (76.42% (SD=0.12)).

Figure 8(c) shows the interaction effect between Technique and Length. For SRD and InkEqu, a much lower ExpRate was oberved in long formulus compared to the short ones, while MathAssist got similar ExpRate in long and short formulas.

By calculating the ExpRate for all participants in one equation under different techniques, we can compare the performance of three techniques in different formulas. As shown in Figure 9, MathAssist

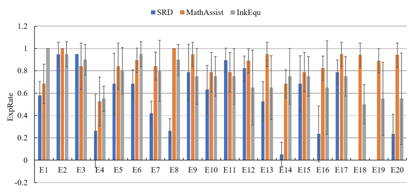


Figure 9: The ExpRate of each mathematical expression among three techniques.

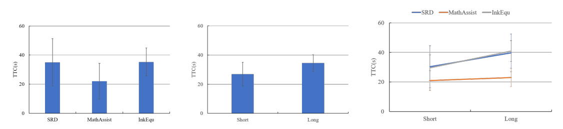
outperformed the other two techniques on most mathematical expression. The reason was that MathAssist can autocomplete results based on partial user input strokes, which helped reduce recognition errors caused by the user inputting strokes of special characters or complex structures. Between SRD and MathAssist, the latter maintained the lead in ExpRate on all formulas, except for E3 and E11. For E3, participants are accustomed to writing the cut-off line first, whereas this line lacking contextual semantics was difficult to recognize, and then led to recommendations based on incorrect results. While E11 contains too many numbers, it was challenging to discover the more qualified expressions in the knowledge base, especially in the case of "log" as in Figure 13(a). Additionally, SRD performed poorly on ExpRate of long formulas, particularly on E14, E18 and E19. The possible cause was that (1) As stated in [62], SRD becomes less accurate as the length of the formulas and the number of symbols increase; (2) these three formulas contain multiple similar-looking ("2" and "z") or special symbols ("!", which is generally recognized as 1 due to a too small or missing dot) that are challenging to recognize correctly ; 3) SRD's weak performance in dense strokes recognition, mainly caused by the participants' choice to write the last part of formulas densely in the input space in order to write long formulas completely. This is also stated in [52] (sometimes users don't leave enough room to write a large formula). MathAssist, in contrast, avoided issues where it was challenging to recognize special or similar-looking symbols found in SRD by autocomplete partial formulas based on partial recognition results or recombined results. Second, long formulas can also give autocompletion more semantic information. In addition, MathAssist provides 5 most likely autocomplete formulas to choose from, and participants could correct the closest formula to get the required if they could not get the exact result from candidates. These were the reasons why MathAssist's performance on complex formulas had not deteriorated. Between MathAssist and InkEqu, MathAssist also maintained the lead in ExpRate on all formulas, except for E1, E3, E4 E6, E14. Notably, there is very little difference on ExpRate for E3 and E4 between the two methods. Since E1 contains only 3 characters and a pair of brackets, MathAssist is unable to accurately recommend subsequent formulas with very little input (perhaps just the strokes of 'n'). Additionally, misrecognized result within the first half of the brackets can also impact subsequent autocomplete. For E6 and E14, InkEqu can select and correct the symbol of similar-look ("2"), special symbols ("θ") and misrecognized symbol of scribbled strokes ("log") by provided candidates. However, MathAssist can only autocomplete the formulas based on the recognition of these symbols, which may be incorrect.

6.2 Time To Completion

The result of TTC is presented in Figure 10. Repeated-measure ANOVA showed a significant main effect of Technique (F(2,14)=34.477, p<0.001, partial η^2=0.683) and Length (F(1,15)=40.961, p<0.01, partial η^2=0.719). Interaction effect between Technique and Length was also significant (F(2,14)=10.712, p<0.001, partial η^2=0.401).

As shown in Figure 10(a), MathAssist got a lowest average TTC of 22.01s (SD=12.34), followed by 35.01s (SD=16.34) for SRD and 35.26s (SD=9.61) for InkEqu. Post-hoc tests showed that the differences between MathAssist and SRD and between MathAssist and InkEqu are all significant (all p<0.001), but no statistical difference between InkEqu and SRD (p=1.00). As shown in Figure 10(b), participants spent more time in editing long formulas (34.54s (SD=5.67)) compared to short formulas (26.97s (SD=8.12)) as expected.

Figure 10(c) shows the interaction effect between Technique and Length. For SRD and InkEqu, their TTCs show essentially the same pattern of change, that is, they both increase rapidly with increasing formula lengths. In contrast, MathAssist's TTC remains relatively constant on both short and long formulas.



(a) TTC under three techniques (b) TTC at two length levels (c) TTC of the three techniques at two length levels

Figure 10: Comparison of all formulas' TTC.

Figure 11 shows the TTC in each equation under different techniques. The results show that of the three techniques, MathAssist takes the least time for each formula, with the exceptions of E1, E3, and E12, which take longer than SRD, and E8, which take longer than InkEqu. For E1 and E12, the subjects' different writing orders in fractional structure affected the recommendation of MathAssist. For E1, which has few symbols and handwritten strokes, the participants would save more time by writing directly than by correcting the wrong recommendation. For E8, it contains the special character "!" in a very short formulas, which makes it difficult for SRD to recognize and results in poor performance for MathAssist. This effect is also reflected in the increased TTC spent on E8 by both SRD and MathAssist. Additionally, We delineate the sources of TTC in detail to derive where delays exist in MathAssist. MathAssist's TTC consists of four parts: writing, recognition, recommendation, and interaction. We also report detailed results for these four parts, where the average writing time was 4.03s (SD=1.63), accounting for 18.33% of TTC; the average recognition time was 10.60s (SD=4.80) and accounting for 48.17% of TTC; the average recommendation time was 0.75s (SD=0.38), accounting for 3.42% of TTC; and the average interaction time was 6.62s (SD=2.67), accounting for 30.08% of TTC. It was important to note that MathAssist recognized formulas once every 1172.03ms on average. The average number of formula identifications per formula was 8.94 (SD=2.88), ranging from 2 to 14 depending on the length of the formulas. As a result of recommendation being a subtask that comes after recognition, the

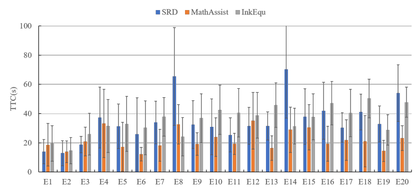


Figure 11: The TTC of each mathematical expression among the three techniques.

frequency was consistent with recognition. Thus, the time per recommendation was 84.21ms, which maintained a very low latency. Last but not least, it should be mentioned that the time delay of our method originates from the recognition and recommendation, which took an average of 1.25s (1256.24 ms) each, of which 6.7% was for recommendation. The low recommendation latency also reflects the fact that the choice of dataset has little impact on the total latency of MathAssist.

6.3 Qualitative evaluation

We used a paired t-test to analysis the post-questionnaire data as the data was normally distributed. As shown in Figure 12, the paired t-test showed significant difference on all six metrics between MathAssist and InkEqu, and Between MathAssist and SRD (p<0.05). The t-value are shown in Table 3. Between SRD and InkEqu, there were also no significant difference on accuracy (t=-0.114, p=0.911), efficiency(t=-0.52, p=0.959),satisfaction(t=0.407, p=0.689),perference(t=0.049, p=0.962) between SRD and InkEqu, but there were significant difference on helpfulness(t=-2.762,p<0.05) and cognitive load (t=9.935, p<0.05). Also, results of the showed that the participants gave MathAssist positive feedback. They thought it is reasonable to add autocomplete in MathAssist (Mean=4.6, SD=0.59), and could help them understand their input (Mean=3.88, SD=0.75). On the question of whether to introduce troublesome, participants rated it relatively low (Mean=3.24, SD=0.72).

Table 3: The t-values on all six metrics between MathAssist and InkEqu, and between MathAssist and SRD

Item	Technique	MathAssist	
		t	p
Accuracy	SRD	-10.182	.000
	InkEqu	3.871	.001
Efficiency	SRD	-11.662	.000
	InkEqu	8.732	.000
Helpfulness	SRD	-9.200	.000
	InkEqu	3.169	.006
Satisfaction	SRD	-4.747	.000
	InkEqu	2.967	.009
Preference	SRD	-6.061	.000
	InkEqu	2.880	.011
Cognitive Load	SRD	4.226	.001
	InkEqu	2.167	.046

In the semi-structured interviews, participants dished out what they saw as the troubles, such as *"I saw the correct result when I was*

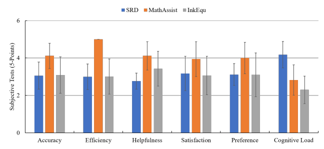


Figure 12: The subjective evaluation among the three techniques (when it comes to cognitive load, lower scores indicate that the method is superior).

writing, but it was replaced when I was ready to select it"(P9), and *"I wished there were more recommended options"* (P10). The reason why the result is replaced was that autocomplete result would be updated again based on the new input strokes. If the user abandoned the full input of the current stroke because he glimpsed the correct result, this would affect both the recognition result and the autocomplete result. As for recommended options, we provide users with five alternative options, which need to be selected by touch up/down within limited user interface. However, users preferred to select some options presented directly on the UI during experiment.

In general, the participants acknowledged that the MathAssist performed better than the other two techniques based on HMER in terms of accuracy and TTC. Between SRD and MathAssist, most participants acknowledged that autocomplete in MathAssist brought better accuracy and efficiency in its role as an assistant than relying solely on recognition: *"as more strokes were input, the autocomplete result would become more accurate"* (P1), *"thanks to autocomplete, it effectively improved the accuracy and efficiency of my expressions input"* (P5). P17 confirmed the benefit of our autocomplete with structural block with *"I felt very advanced that MathAssist could autocomplete some symbols and strokes of the denominator when I finished the numerator"* and P15 thought it was reasonable to introduce autocomplete technique, and stated that *"the worst case was not to accept the autocomplete result, I think I could live with that"*. Some participants also wanted MathAssist to introduce the "select and correct" of InkEqu: *The 'select and correct' provided by InkEqu is convenient for correct symbols like 'θ' and '!'. I hope MathAssist can introduce it"*(P2), *"MathAssist should provide some ability to edit the recognition results directly, so that it is easy to resolve recognition errors for symbols like 'log', 'z', and 'θ'"*(p8). Compared to InkEqu, participants preferred MathAssist. they states *"I like MathAssist because it allows me to write fewer formulas"* (p8) and *"InkEqu is better than SRD, but not as good as MathAssist"*(p10). Of course, there was a downside to introducing a new novelty: *"sometimes, there were grammatical errors in autocomplete results"* (P5, P6), *"the autocomplete result was sometimes delayed, it didn't appear until I started input"* (P16). As well as some expectations of AI: *"I wish MathAssist could self-correct ambiguous symbols in expressions"* (P14) and *"I hope MathAssist could understand my behavior. "It should know what I intended when it encountered symbols that I had previously written several times"* (P10).

Through a narrow opening in MathAssist, we found that participants think it is a trend for AI techniques like autocomplete to better assist humans. such as *"tasks that previously needed human memory, time, or talent could partly be solved by AI."* (P3), *"autocomplete could provide us with more flexible choices and prevent us from making poor decisions in the blind spot of knowledge"* (P7).

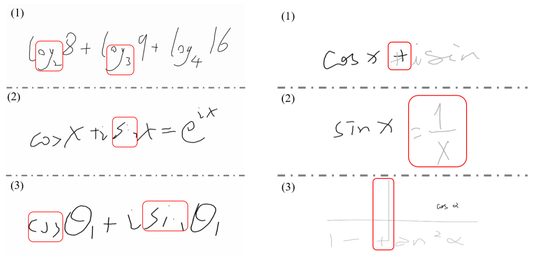
7 DISCUSSION

This paper is the first attempt to propose an autocomplete technique for mathematical expressions. We formalized the mathematical expression with an inorder tree and solved the two problems of tree matching and structural block recommendation in the automatic completion process. The advantages of doing so include: 1) converting a flat string with obfuscated structure into an inorder tree, which makes expressions' structure more explicit; 2) with the inorder tree, we can use tree kernels to compare the similarity between the structures of two mathematical expressions; and 3) recommending symbol or structural blocks in the most relevant tree by search subpath allows a more flexible way to write formula. Compared to the two baseline methods, MathAssist enabled participants to obtain more precise formulas in less time. Following are answers to the three research questions of this study:

- `To RQ1`: MathAssist is 32.78% and 10.55% higher than SRD and InkEqu in ExpRate. Additionally, MathAssist is 37.14% and 37.58% faster than SRD and InkEqu on TTC.
- `To RQ2`: The TTC of SRD and InkEqu increases with formula length while ExpRate of SRD and InkEqu decreases with formula length. TTC and ExpRate of MathAssist remained relatively constant under different formula lengths.
- `To RQ3`: MathAssist was rated higher on accuracy, efficiency, helpfulness, satisfaction, and preference compared to SRD and InkEqu, while MathAssist performed poorly on cognitive load compared to InkEqu (higher ratings indicate a greater cognitive load required by the participants). The subjects also praised MathAssist's autocomplte technique in the semi-structured interviews.

7.1 Possible influence on MathAssist

There are a few potential causes influencing mathAssist's performance, 1) Incorrect recognition outcomes may result in poor performance of autocomplete. As shown in Figure 13(a), three items ($\log_2$, sin, cos) written by a participants are not correctly recognized by recognition module, which prevents MathAssist from appropriately recommending the mathematical expressions including these three items (e.g., E11, E7). 2) Delayed results could interrupts users' writing and thinking. As shown in Figure 13(b), an autocomplete result for "$\cos x$" is provided only after the participant writes down the subsequent symbol "+" for "$\cos x$". In the worst cases, the recognition can take as long as 2000 ms to complete. 3) It was possible to autocomplete unreasonable mathematical expressions. As shown in Figure 13(b), the autocomplete result $\frac{1}{x}$ does not quite fit the logic of the trigonometric function. The reason is that MathAssist did not consider the semantic information in the current structure. For instance, the mathematical expression "$\sin x$" should be automatically completed with "$+ \cos x$" rather than "$\frac{1}{x}$". 4) Some autocomplete strokes were not displayed correctly. As shown in Figure 13(b), the strokes of "t" in *tan* extends vertically into the numerator strokes. This error may be caused by improper strokes coordinates in database. To address the above four issues, we may need to improve the efficiency and accuracy of the recognition module, add domain knowledge to the system, or enlarge the mathematical expression dataset.



(a) Errors caused by personalized writing. (b) Some examples where MathAssist is difficult to be competent.

Figure 13: Possible errors or deficiencies in MathAssist.

The performance of our approaches may also be impacted by the unfriendly display of autocomplete results. We found that when the current autocomplete result is not the one the participants want, they sometimes choose to continue writing and easily ignore other alternative results listed above the input widget. As participants commented, *"sometimes I overlook checking the other alternative autocomplete results"*, *"the alternative formulas are far away from my input strokes, which affects the visual experience"*, *"recommended tips are best displayed in a centralized view"*, and *"in the case that the recommended strokes are not reasonable, I may continue to write and thus forget to check other alternative results"*. While not to limit participants' free writing, the input widget takes up more space in the UI, which may force us to provide fewer candidate results and far from the center of the input widget, making it easier for participants to abandon the selection of candidate results during the process. This could pose a problem in how to provide more optional results in a constrained space, especially for results that require more room, like standard typesetting of mathematical expressions.

Finally, the choice of the core algorithm will also have an impact on MathAssist. The primary reason for choosing SRD as the core algorithm over the current best USTC-iFLYTEK[32] is its superior recognition accuracy among the current open-source methods, making it convenient for us to use for reproduction and reuse. Additionally, USTC-iFLYTEK utilizes data augmentation in the training phase to alleviate the problem of limited training data[32].

7.2 Reasonableness of the MathAssist

One point worth discussing about the results recommended by MathAssist is that why could the recommended result be the formula that the user intends to write? We attribute this as follows: Firstly, by obtaining the same or more similar outcomes to the current input from semantically relevant mathematical expressions in the database, MathAssist provides five potential results: Second, our experiments currently are only focused on the 4 types of expression mentioned before, as they can infer some semantic links from expression alone; And most importantly, to prevent semantically irrelevant results, we use structural recommendations as the core in the case of associating symbols in the input. Another issue worth discussing is whether MathAssist will remain helpful if the accuracy of the basic recognition method improves. We believe this concern is unnecessary. First, despite the fact that HMER can overcome problems brought on by similar symbols and complicated structures, it is challenging to completely eliminate errors resulting from human differences. Second, the performance of MathAssist will be better based on the great accuracy of its basic method, which can avoid tedious writing and shorten the completion time.

7.3 Findings

We observed that participants spent a relatively short amount of time when using the SRD technique to write E18 and E19 while achieving zero recognition accuracy on these two formulas. We suspect that participants may have lost patience when using the SRD technique to write these two formulas. In contrast, technologies with higher recognition accuracy (i.e., MathAssist and InkEqu) did not exhibit this phenomenon. This may indicate that the algorithm's recognition accuracy influences participants' confidence. Participants have more confidence in an algorithm with higher recognition accuracy; conversely, participants could lose confidence in algorithms with lower recognition accuracy, leading them to be unwilling to spend time repeatedly modifying their input.

Some interesting things was also found. A few participants attempt to adapt to MathAssist by discovering rules of cooperation. such as *"I think it can recognize more accurately when I write larger symbols"* or *"I should not write in cursive handwriting"*. Result-wise, it is a positive that participants go to adapt MathAssist, but it is worth investigating whether it is advantageous for autocomplete technique to change user behavior. Because some participants worried that MathAssist would breed laziness and make them unwilling to think. So, what kind of assistance should assistive HMER technique provide to humans, and how the role of humans in technique-assisted cooperation should be transformed, is an open ultimate proposition that should be explored in future research.

We also found that the participants had exorbitant expectation about MathAssist. In most cases, participants expected that MathAssist would have limitless powers, such as correctly recognizing their scribbled strokes or supporting them create formulas, to help them acquire the results they needed. Participants also expressed a desire for MathAssist to autonomously raise its autocomplete ability with continued use. Furthermore, participants provided some fresh advice about our technique and possible requirements in mathematical expression input application, such as personalized autocomplete for formula strokes, automatic deduction and calculation. Those findings gave us direction for enhancing the capabilities of our technique.

8 LIMITATIONS AND FUTURE WORK

Our work also has some limitations. In our study, considering participant fatigue, we chose only four categories, totaling 20 mathematical formulas, as the test bed. The performance of MathAssist on mathematical expressions involving other categories, such as matrices and set operations, has not been verified. Second, SRD is the only basic method we employed for the recognition module. We do not validate other candidate HMER methods on recognition module which could potentially further improve MathAssist's performance. Third, we did not assess how well MathAssist performed on the free creation of mathematical expression, which may limit the capabilities that MathAssist delivers in practice. Furthermore, the basic method SRD used in this paper is trained on the CHROME public dataset, which includes only 101 mathematical symbols. This limitation results in MathAssist supporting autocompletion for a relatively smaller number of symbols. In future work, we will introduce alternative HMER methods in the recognition module to improve the reliability of our autocomplete. We are also interested in evaluating MathAssist in term of free writing of mathematical expressions.

9 CONCLUSION

In this paper, we present MathAssist, a technique that autocomplete mathematical expressions based on partial users' input strokes by using a tree-based mathematical expression formalization, a improved tree kernel method, and a structural block recommendation. In our evaluation, participants who used MathAssist were able to obtain mathematical expressions more accurately and efficiently. In addition, MathAssist received positive feedback from subjective evaluations. Participants believed that it was beneficial to introduce autocomplete technique into mathematical expression recognition. We also found some interesting issues that demand further research, such as exploring how MathAssist would be affected by enhanced recognition capabilities of the recognition module.

ACKNOWLEDGMENTS

We thank anonymous reviewers for their helpful comments and suggestions. This work was supported by the National Natural Science Foundation of China (Grant no. 62172397) and Youth Innovation Promotion Association CAS (Grant no. 2020113).

REFERENCES

[1] Dragan Ahmetovic, Cristian Bernareggi, Marco Bracco, Nadir Murru, Tiziana Armano, and Anna Capietto. 2021. LaTeX as an Inclusive Accessibility Instrument for Highschool Mathematical Education. In *Proceedings of the 18th International Web for All Conference* (Ljubljana, Slovenia) *(W4A '21)*. Association for Computing Machinery, New York, NY, USA, Article 14, 9 pages. https://doi.org/10.1145/3430263.3452444

[2] Francisco Álvaro, Joan-Andreu Sánchez, and José-Miguel Benedí. 2014. Recognition of On-Line Handwritten Mathematical Expressions Using 2D Stochastic Context-Free Grammars and Hidden Markov Models. *Pattern Recogn. Lett.* 35 (jan 2014), 58–67. https://doi.org/10.1016/j.patrec.2012.09.023

[3] Francisco Álvaro, Joan-Andreu Sánchez, and José-Miguel Benedí. 2016. An Integrated Grammar-Based Approach for Mathematical Expression Recognition. *Pattern Recogn.* 51, C (mar 2016), 135–147. https://doi.org/10.1016/j.patcog.2015.09.013

[4] Francisco Alvaro and Richard Zanibbi. 2013. A Shape-Based Layout Descriptor for Classifying Spatial Relationships in Handwritten Math. In *Proceedings of the 2013 ACM Symposium on Document Engineering* (Florence, Italy) *(DocEng '13)*. Association for Computing Machinery, New York, NY, USA, 123–126. https://doi.org/10.1145/2494266.2494315

[5] Rahul Amlekar, Andrés Felipe Rincón Gamboa, Keheliya Gallaba, and Shane McIntosh. 2018. Do Software Engineers Use Autocompletion Features Differently than Other Developers?. In *Proceedings of the 15th International Conference on Mining Software Repositories* (Gothenburg, Sweden) *(MSR '18)*. Association for Computing Machinery, New York, NY, USA, 86–89. https://doi.org/10.1145/3196398.3196471

[6] Lisa Anthony, Jie Yang, and Kenneth R. Koedinger. 2005. Evaluation of Multimodal Input for Entering Mathematical Equations on the Computer. In *CHI '05 Extended Abstracts on Human Factors in Computing Systems* (Portland, OR, USA) *(CHI EA '05)*. Association for Computing Machinery, New York, NY, USA, 1184–1187. https://doi.org/10.1145/1056808.1056872

[7] Lisa Anthony, Jie Yang, and Kenneth R. Koedinger. 2012. A Paradigm for Handwriting-based Intelligent Tutors. *International Journal of Human-Computer Studies* 70, 11 (2012), 866–887. https://doi.org/10.1016/j.ijhcs.2012.04.003

[8] Ahmad-Montaser Awal, Harold Mouchre, and Christian Viard-Gaudin. 2014. A Global Learning Approach for an Online Handwritten Mathematical Expression Recognition System. *Pattern Recogn. Lett.* 35, C (jan 2014), 68–77.

[9] Xiaohang Bian, Bo Qin, Xiaozhe Xin, Jianwu Li, Xuefeng Su, and Yanfeng Wang. 2022. Handwritten Mathematical Expression Recognition via Attention Aggregation based Bi-directional Mutual Learning. In *AAAI*.

[10] Andrea Bunt, Michael Terry, and Edward Lank. 2013. Challenges and opportunities for Mathematics Software in Expert Problem Solving. *Human–Computer Interaction* 28, 3 (2013), 222–264.

[11] Xueke Chi, Da-Han Wang, Yuefeng Wu, and Yun Wu. 2021. Handwritten Mathematical Expression Recognition with Self-Attention. In *2021 4th International Conference on Algorithms, Computing and Artificial Intelligence* (Sanya, China) *(ACAI'21)*. Association for Computing Machinery, New York, NY, USA, Article 69, 6 pages. https://doi.org/10.1145/3508546.3508615

[12] Microsoft Community. August 17, 2018. *what is the math input panel about?* Retrieved October 5, 2023 from https://answers.microsoft.com/en-us/windows/forum/all/what-is-the-math-input-panel-about/ce63157b-f97a-4939-8145-f0a2a3feb4e6

[13] Daniela S. Costa, Carlos A. B. Mello, and Marcelo d'Amorim. 2021. A Comparative Study on Methods and Tools for Handwritten Mathematical Expression Recognition. In *Proceedings of the 21st ACM Symposium on Document Engineering* (Limerick, Ireland) *(DocEng '21)*. Association for Computing Machinery, New York, NY, USA, Article 26, 4 pages. https://doi.org/10.1145/3469096.3474936

[14] Haisong Ding, Kai Chen, and Qiang Huo. 2021. An Encoder-Decoder Approach to Handwritten Mathematical Expression Recognition with Multi-head Attention and Stacked Decoder. In *Document Analysis and Recognition – ICDAR 2021*, Josep Lladós, Daniel Lopresti, and Seiichi Uchida (Eds.). Springer International Publishing, Cham, 602–616.

[15] Wendy Doubé and Jeanie Beh. 2012. Typing over Autocomplete: Cognitive Load in Website Use by Older Adults. In *Proceedings of the 24th Australian Computer-Human Interaction Conference* (Melbourne, Australia) *(OzCHI '12)*. Association for Computing Machinery, New York, NY, USA, 97–106. https://doi.org/10.1145/2414536.2414553

[16] Yingnan Fu, Tingting Liu, Ming Gao, and Aoying Zhou. 2020. EDSL: An Encoder-Decoder Architecture with Symbol-Level Features for Printed Mathematical Expression Recognition. *ArXiv* abs/2007.02517 (2020).

[17] MohammadTaghi Hajiaghayi, Saeed Seddighin, and Xiaorui Sun. 2019. Massively Parallel Approximation Algorithms for Edit Distance and Longest Common Subsequence *(SODA '19)*. Society for Industrial and Applied Mathematics, USA, 1654–1672.

[18] Xiyuan He and Zhuohao Zhang. 2019. GPK: An Efficient Special Symbol Input Method for Keyboards. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI EA '19)*. Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3290607.3308457

[19] Ken Hinckley, Koji Yatani, Michel Pahud, Nicole Coddington, Jenny Rodenhouse, Andy Wilson, Hrvoje Benko, and Bill Buxton. 2010. Pen + Touch = New Tools *(UIST '10)*. Association for Computing Machinery, New York, NY, USA, 27–36. https://doi.org/10.1145/1866029.1866036

[20] Chen-Yuan Hsu, Li-Yi Wei, Lihua You, and Jian Jun Zhang. 2020. Autocomplete Element Fields. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3313831.3376248

[21] Thanapong Intharah, Michael Firman, and Gabriel J. Brostow. 2018. RecurBot: Learn to Auto-Complete GUI Tasks From Human Demonstrations. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI EA '18)*. Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3170427.3188532

[22] Wenhui Kang, Jin Huang, Feng Tian, Xiangmin Fan, Jie Liu, and Guozhong Dai. 2021. Human-in-the-Loop Based Online Handwriting Mathematical Expressions Recognition. *Journal of Computer-Aided Design & Computer Graphics* 33, 11 (2021), 1773–1785. https://doi.org/10.3724/SP.J.1089.2021.18796

[23] Giovanni Yoko Kristianto, Goran Topić, and Akiko Aizawa. 2017. Utilizing Dependency Relationships between Math Expressions in Math IR. *Information Retrieval Journal* 20, 2 (2017), 132–167.

[24] Anh Duc Le, Bipin Indurkhya, and Masaki Nakagawa. 2019. Pattern Generation Strategies for Improving Recognition of Handwritten Mathematical Expressions. *Pattern Recogn. Lett.* 128, C (dec 2019), 255–262. https://doi.org/10.1016/j.patrec.2019.09.002

[25] Anh Duc Le and Masaki Nakagawa. 2017. Speedup of Parsing for Recognition of Online Handwritten Mathematical Expressions. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, Vol. 01. 896–901. https://doi.org/10.1109/ICDAR.2017.151

[26] Anh Duc Le, Hai Dai Nguyen, Bipin Indurkhya, and Masaki Nakagawa. 2019. Stroke Order Normalization for Improving Recognition of Online Handwritten

Mathematical Expressions. 22, 1 (mar 2019), 29–39. https://doi.org/10.1007/s10032-019-00315-2

[27] Minxuan Li. 2019. An Improved FCM Clustering Algorithm Based on Cosine Similarity. In *Proceedings of the 2019 International Conference on Data Mining and Machine Learning* (Hong Kong, Hong Kong) *(ICDMML 2019)*. Association for Computing Machinery, New York, NY, USA, 103–109. https://doi.org/10.1145/3335656.3335693

[28] Zhe Li, Lianwen Jin, Songxuan Lai, and Yecheng Zhu. 2020. Improving Attention-Based Handwritten Mathematical Expression Recognition with Scale Augmentation and Drop Attention. In *2020 17th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. 175–180. https://doi.org/10.1109/ICFHR2020.2020.00041

[29] Jo-Yu Lo, Da-Yuan Huang, Tzu-Sheng Kuo, Chen-Kuo Sun, Jun Gong, Teddy Seyed, Xing-Dong Yang, and Bing-Yu Chen. 2019. AutoFritz: Autocomplete for Prototyping Virtual Breadboard Circuits. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3290605.3300633

[30] Mollie MacGregor. 1989. Reading and Writing Mathematics. *Australian Journal of Reading* 12, 2 (1989), 153–227.

[31] Mahshad Mahdavi and Richard Zanibbi. 2019. Tree-based Structure Recognition Evaluation for Math Expressions: Techniques and Case Study. In *Proc. IAPR Int. Work. Graph. Recognit.*

[32] Mahshad Mahdavi, Richard Zanibbi, Harold Mouchere, Christian Viard-Gaudin, and Utpal Garain. 2019. ICDAR 2019 CROHME + TFD: Competition on Recognition of Handwritten Mathematical Expressions and Typeset Formula Detection. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*. 1533–1538. https://doi.org/10.1109/ICDAR.2019.00247

[33] Mahshad Mahdavi, Richard Zanibbi, Harold Mouchère, Christian Viard-Gaudin, and Utpal Garain. 2019. ICDAR 2019 CROHME + TFD: Competition on Recognition of Handwritten Mathematical Expressions and Typeset Formula Detection. In *2019 International Conference on Document Analysis and Recognition, ICDAR 2019, Sydney, Australia, September 20-25, 2019*. IEEE, 1533–1538. https://doi.org/10.1109/ICDAR.2019.00247

[34] Anne Mangen, Liss G Anda, Gunn H Oxborough, and Kolbjørn Brřnnick. 2015. Handwriting Versus Keyboard Writing: Effect on Word Recall. *Journal of writing research* 7, 2 (2015), 227–247.

[35] S Manikandan. 2010. Data Transformation. *Journal of Pharmacology and Pharmacotherapeutics* 1, 2 (2010), 126.

[36] Alexandra Mendes, Roland Backhouse, and Joao F. Ferreira. 2014. Structure Editing of Handwritten Mathematics: Improving the Computer Support for the Calculational Method. In *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces* (Dresden, Germany) *(ITS '14)*. Association for Computing Machinery, New York, NY, USA, 139–148. https://doi.org/10.1145/2669485.2669495

[37] Microsoft. 2022. *Create Math Equations Using Ink or Text with Math Assistant in OneNote*. Retrieved November 26, 2022 from https://support.microsoft.com/en-us/topic/create-math-equations-using-ink-or-text-with-math-assistant-in-onenote-dc818fad-60e0-432d-8cae-b61f9febf874

[38] Yuma Nagao and Nobutaka Suzuki. 2017. Classification of MathML Expressions Using Multilayer Perceptron. In *Proceedings of the 2017 ACM Symposium on Document Engineering* (Valletta, Malta) *(DocEng '17)*. Association for Computing Machinery, New York, NY, USA, 133–136. https://doi.org/10.1145/3103010.3121026

[39] Tomoyasu Nakano, Yuki Koyama, Masahiro Hamasaki, and Masataka Goto. 2019. Autocomplete Vocal-Fo Annotation of Songs Using Musical Repetitions. In *Proceedings of the 24th International Conference on Intelligent User Interfaces: Companion* (Marina del Ray, California) *(IUI '19)*. Association for Computing Machinery, New York, NY, USA, 71–72. https://doi.org/10.1145/3308557.3308700

[40] Cuong Tuan Nguyen, Hung Tuan Nguyen, Kei Morizumi, and Masaki Nakagawa. 2021. Temporal Classification Constraint for Improving Handwritten Mathematical Expression Recognition. In *Document Analysis and Recognition – ICDAR 2021 Workshops*, Elisa H. Barney Smith and Umapada Pal (Eds.). Springer International Publishing, Cham, 113–125.

[41] Cuong Tuan Nguyen, Thanh-Nghia Truong, Hung Tuan Nguyen, and Masaki Nakagawa. 2021. Global Context for Improving Recognition of Online Handwritten Mathematical Expressions. Springer-Verlag, Berlin, Heidelberg, 617–631. https://doi.org/10.1007/978-3-030-86331-9_40

[42] Mengqi Peng, Li-yi Wei, Rubaiat Habib Kazi, and Vladimir G. Kim. 2020. *Autocomplete Animated Sculpting*. Association for Computing Machinery, New York, NY, USA, 760–777. https://doi.org/10.1145/3379337.3415884

[43] Mengqi Peng, Jun Xing, and Li-Yi Wei. 2018. Autocomplete 3D Sculpting. *ACM Trans. Graph.* 37, 4, Article 132 (jul 2018), 15 pages. https://doi.org/10.1145/3197517.3201297

[44] Ronald E. Robertson, Shan Jiang, David Lazer, and Christo Wilson. 2019. Auditing Autocomplete: Suggestion Networks and Recursive Algorithm Interrogation. In *Proceedings of the 10th ACM Conference on Web Science* (Boston, Massachusetts, USA) *(WebSci '19)*. Association for Computing Machinery, New York, NY, USA, 235–244. https://doi.org/10.1145/3292522.3326047

[45] Sandip Roy, Rajesh Bose, and Debabrata Sarddar. 2015. Non-recursive Inorder Traversal on Constructed Threaded K-D Tree for Efficient Cloud Based Space Partitioning. In *2015 International Conference on Advances in Computer Engineering and Applications*. 665–668. https://doi.org/10.1109/ICACEA.2015.7164774

[46] Sakshi, Shivani Gautam, Chetan Sharma, and Vinay Kukreja. 2021. Handwritten Mathematical Symbols Classification Using WEKA. In *Applications of Artificial Intelligence and Machine Learning*, Ankur Choudhary, Arun Prakash Agrawal, Rajasvaran Logeswaran, and Bhuvan Unhelkar (Eds.). Springer Singapore, Singapore, 33–41.

[47] Sakshi, Chetan Sharma, and Vinay Kukreja. 2022. The Survey on Handwritten Mathematical Expressions Recognition. In *Cyber Intelligence and Information Retrieval*, João Manuel R. S. Tavares, Paramartha Dutta, Soumi Dutta, and Debabrata Samanta (Eds.). Springer Singapore, Singapore, 129–135.

[48] A. Sapounakis, I. Tasoulas, and P. Tsikouras. 2006. Ordered Trees and the Inorder Traversal. *Discrete Math.* 306, 15 (aug 2006), 1732–1741. https://doi.org/10.1016/j.disc.2006.03.044

[49] Guangcun Shan, Hongyu Wang, and Wei Liang. 2020. Robust Encoder-Decoder Learning Framework for Offline Handwritten Mathematical Expression Recognition Based on a Multi-scale Deep Neural Network. *Science China Information Sciences* 64 (2020), 1–3.

[50] Kilho Shin and Tetsuji Kuboyama. 2014. A Comprehensive Study of Tree Kernels. In *New Frontiers in Artificial Intelligence*, Yukiko Nakano, Ken Satoh, and Daisuke Bekki (Eds.). Springer International Publishing, Cham, 337–351.

[51] Jun Tang, Qing Wu, and Yinghong Li. 2015. An Optimization Algorithm of Chinese Word Segmentation Based on Dictionary *(ICNISC '15)*. IEEE Computer Society, USA, 259–262. https://doi.org/10.1109/ICNISC.2015.109

[52] Eugene M. Taranta and Joseph J. LaViola. 2015. Math Boxes: A Pen-Based User Interface for Writing Difficult Mathematical Expressions. In *Proceedings of the 20th International Conference on Intelligent User Interfaces* (Atlanta, Georgia, USA) *(IUI '15)*. Association for Computing Machinery, New York, NY, USA, 87–96. https://doi.org/10.1145/2678025.2701400

[53] Thanh-Nghia Truong, Cuong Tuan Nguyen, and Masaki Nakagawa. 2022. Syntactic Data Generation for Handwritten Mathematical Expression Recognition. *Pattern Recogn. Lett.* 153, C (jan 2022), 83–91. https://doi.org/10.1016/j.patrec.2021.12.002

[54] Thanh-Nghia Truong, Cuong Tuan Nguyen, Khanh Minh Phan, and Masaki Nakagawa. 2020. Improvement of End-to-End Offline Handwritten Mathematical Expression Recognition by Weakly Supervised Learning. In *2020 17th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. 181–186. https://doi.org/10.1109/ICFHR2020.2020.00042

[55] Jiaming Wang, Jun Du, Jianshu Zhang, Bin Wang, and Bo Ren. 2021. Stroke Constrained Attention Network for Online Handwritten Mathematical Expression Recognition. *Pattern Recognition* 119 (2021), 108047. https://doi.org/10.1016/j.patcog.2021.108047

[56] Francis R Willett, Donald T Avansino, Leigh R Hochberg, Jaimie M Henderson, and Krishna V Shenoy. 2021. High-Performance Brain-to-Text Communication via Handwriting. *Nature* 593, 7858 (2021), 249–254.

[57] Jacob O. Wobbrock, Leah Findlater, Darren Gergle, and James J. Higgins. 2011. The Aligned Rank Transform for Nonparametric Factorial Analyses Using Only Anova Procedures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI '11)*. Association for Computing Machinery, New York, NY, USA, 143–146. https://doi.org/10.1145/1978942.1978963

[58] Jin-Wen Wu, Fei Yin, Yan-Ming Zhang, Xu-Yao Zhang, and Cheng-Lin Liu. 2020. Handwritten Mathematical Expression Recognition via Paired Adversarial Learning. *Int. J. Comput. Vision* 128, 10–11 (nov 2020), 2386–2401. https://doi.org/10.1007/s11263-020-01291-5

[59] Jun Xing, Hsiang-Ting Chen, and Li-Yi Wei. 2014. Autocomplete Painting Repetitions. *ACM Trans. Graph.* 33, 6, Article 172 (nov 2014), 11 pages. https://doi.org/10.1145/2661229.2661247

[60] Ye Yuan, Xiao Liu, Wondimu Dikubab, Hui Liu, Zhilong Ji, Zhongqin Wu, and Xiang Bai. 2022. Syntax-Aware Network for Handwritten Mathematical Expression Recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 4553–4562.

[61] Jianshu Zhang, Jun Du, and Lirong Dai. 2019. Track, Attend, and Parse (TAP): An End-to-End Framework for Online Handwritten Mathematical Expression Recognition. *Trans. Multi.* 21, 1 (jan 2019), 221–233. https://doi.org/10.1109/TMM.2018.2844689

[62] Jianshu Zhang, Jun Du, Yongxin Yang, Yi-Zhe Song, and Lirong Dai. 2021. SRD: A Tree Structure Based Decoder for Online Handwritten Mathematical Expression Recognition. *Trans. Multi.* 23 (jan 2021), 2471–2480. https://doi.org/10.1109/TMM.2020.3011316

[63] Jianshu Zhang, Jun Du, Shiliang Zhang, Dan Liu, Yulong Hu, Jinshui Hu, Si Wei, and Lirong Dai. 2017. Watch, Attend and Parse: An End-to-End Neural Network Based Approach to Handwritten Mathematical Expression Recognition. *Pattern Recognition* 71 (2017), 196–206. https://doi.org/10.1016/j.patcog.2017.06.017

[64] Ting Zhang, Harold Mouchère, and Christian Viard-Gaudin. 2020. A Tree-BLSTM-Based Recognition System for Online Handwritten Mathematical Expressions. *Neural Comput. Appl.* 32, 9 (may 2020), 4689–4708. https://doi.org/10.1007/s00521-

018-3817-2

[65] Dmytro Zhelezniakov, Anastasiia Cherneha, Viktor Zaytsev, Tetiana Ignatova, Olga Radyvonenko, and Oleg Yakovchuk. 2020. Evaluating New Requirements to Pen-Centric Intelligent User Interface Based on End-to-End Mathematical Expressions Recognition. In *Proceedings of the 25th International Conference on Intelligent User Interfaces* (Cagliari, Italy) *(IUI '20)*. Association for Computing Machinery, New York, NY, USA, 212–220. https://doi.org/10.1145/3377325.3377482

[66] Dmytro Zhelezniakov, Viktor Zaytsev, and Olga Radyvonenko. 2021. Online Handwritten Mathematical Expression Recognition and Applications: A Survey. *IEEE Access* 9 (2021), 38352–38373. https://doi.org/10.1109/ACCESS.2021.3063413