Spelling Check: A New Cognition-Inspired Sequence Learning Memory

Thasayu Soisoonthorn [1,*], Herwig Unger [2], and Maleerat Maliyaem [1]

[1] Faculty of Information Technology and Digital Innovation, King Mongkut's University of Technology North Bangkok, Bangkok, Thailand; Email: maleerat.m@itd.kmutnb.ac.th (M.M.)
[2] University of Hagen, Hagen, Germany; Email: herwig.unger@gmail.com (H.U.)
*Correspondence: thasayu@gmail.com (T.S.)

***Abstract*—This study aimed to use a cognition-inspired method following Hawkins's approach to optimize learning sequences for efficiency. The model for this learning approach is a new, flexible associative form of memory that can handle keys of different lengths to address all fitting sequences. Furthermore, it cannot only identify existing sequences but also learn new ones and ensure fault-tolerant operations. After introducing such memory hardware, its practicability is approved as a new kind of spelling checker. The evaluation uses the TREC-5 Confusion Track standard dataset to automatically correct incorrect words by comparing them with Levenshtein Distance, pyspellchecker, Long Short-Term Memory (LSTM), and Semantically Conditioned LSTM plus Elmo Transformer (Elmosclstm). In a small data set and at the word level, the processing time is only 0.001s, which is lower than other methods. At the sentence level, the cognition-inspired method can achieve 99.31% accuracy, better than Elmosclstm at 81.97% for training data. In a big data set and at the word level, the highest accuracy is 87.38% and 87.03%, beyond Elmosclstm at 77.44% and 74.41% for training data and testing data. At the sentence level, the cognition-inspired method can achieve 96.73% and 91.42%, better than Elmosclstm at 81.50% and 72.18% for training and testing data, respectively.**

***Keywords*—Artificial Intelligence (AI), Hierarchical Temporal Memory (HTM), spelling check**

I. Introduction

Research concerning techniques for error detection and correcting spelling errors is a topic in Natural Language Processing (NLP) that has a long and robust history. A spelling error makes text harder to read and process. The spelling check is used to correct errors and gain information values. Many applications in NLP must correct inputs before processing; otherwise, it can impact the result.

At present, applications for spelling checks are utilized in various areas and daily life. For example, interactive spelling correction systems that highlight incorrect words and suggest corrections as well as provide the following words improve the productivity of work with text and provide convenience, especially on mobile devices. They are also used in search engines [1] that provide corrected words if an error is detected in the search query or grammar checking in an office program by identifying errors and providing suggestions so a user can correct them.

In some cases, interactive correction requests a person to interact, which takes time to process. Hence, an automatic spelling correction system is introduced. For medical records used in the diagnosis and treatment process, for example, any error can significantly impact patients, medical research, and organization processes. Furthermore, document digitalization to reduce the use of paper and keep information in a database uses optical character recognition (OCR) to transform images into text. One crucial aspect of the post-processing process is spell-checking because OCR alone cannot provide perfect results.

One difference between automatic spelling correction systems and interactive spelling correction systems is that the second system prompts the system to respond promptly, with no delay for a human to interact.

The rest of the paper is organized as follows. Section II defines related works, some of which are used to evaluate the newly proposed method. Section III provides background and inspiration for the research. Section IV explains the concept of the proposed approach, while Section V describes the implementation in a hardware simulation to serve as a proof of concept by using commodity hardware and evaluation spelling check performance on a standard dataset, TREC-5 Confusion Track [2]. Finally, Section VI and VII summarizes the contributions and the findings.

II. Related Works

The first type in spelling check, called *non-word error*, is a word in incorrect form and not in the dictionary. This type of error can be handled by searching from the dictionary that checks similarity among words such as Levenshtein Distance (LD) [3] or Damerau-Levenshtein Distance (DLD) [4]. LD uses edit, substitution, deletion, and insertion operations, but DLD is the same as LD, except it includes transposition. However, this approach requests computation for every input word to compare

Manuscript received September 8, 2022; revised November 5, 2022; accepted December 14, 2022; published May 5, 2023.

with all words in the dictionary, and processing time increases with the number of words in the dictionary. For example, the Oxford dictionary contains 273,000 words and increases yearly. There are many techniques to improve the search for words, such as a search tree [5] or hash table [6] based on a technique called *Approximate string matching* [7]. The technique is similar to this research as it finds strings that match a pattern approximately rather than exactly. The difference is that it creates the proximity of a matching word as edit distance provides. Another approach was invented by Peter Norvig [8], which takes a word and brute force for all possible edit distances. This method is mainly used as a standard library or pyspellchecker [9] in Python. With default edit distance 2, the word "are" can be revised to create 182 possible words, such as 'rae', 'aer', 'rre', 'ware', etc. The limitation of this approach is that it is a brute force method; it takes time to create all possible words to match a word in the dictionary, and the misspelled word can also be more than two edit distances. Nevertheless, it is similar to this research, using correct words and a dictionary to create a word list. Another approach that can learn from data and error is deep learning, a machine-learning model that commonly uses Long Short-Term Memory (LSTM) [10, 11]. Its approach provides a good result but requests training data for correct and incorrect words. Also, the processing time is high if no GPU is supported.

More than one word is often required. For example, the misspelled word "site" can also be "size" or "side," depending on the sentence. Thus, another *real-world error* type has been introduced. The word is misspelled, but it is still in the dictionary. A common approach to cope with this problem is to use its surrounding context. A common approach uses edit distance as an error model and n-gram for the context model. Deep learning is also used to cope with this problem by using the Sequence to Sequence Learning Model and integrating it with a transformer to improve the model [12]. This research also uses a deep learning model, Neuspell [13], with SC-LSTM plus ELMO (input), or Elmosclstm for comparison.

III. Inspiration

A. The Brain

The concept for this research starts with the brain and its components [14]. The brain is comprised of many distinct functional areas that evolved gradually from invertebrates to mammals. Exactly how the brain works remains a mystery. Even though we have only a partial understanding of how the brain works, we can roughly estimate its functions from research and experiments, including examining problems caused by brain injuries. The following parts are crucial for a particular cognitive function and are the subject of considerable current research. Also, they are the essential parts of the brains used for this research as follows.

1) Hippocampus is a crucial part of the brain that works with memories. The hippocampus receives inputs from the entire neocortex and projects back to the same areas. The memory in the hippocampus is only temporary. It can play back a sequence of events in context and activate the neocortex area that was activated by the event itself. Typically, this playback occurs during REM sleep; the memories stored for a short term activate back to long-term storage in the neocortex areas that were activated during the original episode. The hippocampus allows vertebrates that are older than mammals, such as lizards and birds, to learn from experience, even without the neocortex.

2) Thalamus is the gateway to the neocortex, which is very near the center of the brain by passing information to and from various areas of the neocortex. All senses except the olfactory system have almost the same process, involving some peripheral processing, followed by a projection to a specific area of the thalamus, which then projects to a primary area of the neocortex for that sense. The olfactory system is the only sensory system in which there is a direct projection from the olfactory bulb to the neocortex. However, the neocortex relays it to the thalamus and projects it back to the neocortex. Because the thalamus receives inputs from all the senses and from the motor control system and the reticular formation, which is responsible for alertness and attention, it is like the hub of a wheel that acts as the concentrator and distributor of all forces.

3) Sensory multiple signals are sent from sensory neurons to the brain, allowing us to experience smell, taste, sight, hearing, and touch. There are five perception systems in humans composed of visual, auditory, skin sense, taste, and smell. The brain processes these five sensory inputs to understand the environment and decide the appropriate action in the motor system (movement). These perception systems rely on certain receptors, specialized neural cells that respond to a specified environment and send signals or action potentials to the brain. Even though these receptors receive different types of sensors, the senses are encoded with the same type of information before being sent to the neocortex.

4) The Neocortex is the largest part of the brain, which is what we see when we look at a brain from above or the side. The neocortex is around a 1.5 square-foot sheet of cells wadded up a bit to fit inside the head, accounting for 80 percent of its weight. Intelligent and adaptive behavior in mammals is associated with the neocortex. Five attributes of the neocortex include uniform, Invariant Representation, Hierarchy, and Auto-associated.

B. Hierarchical Temporal Memory (HTM)

HTM [15, 16] is a theory that was described initially in the book "On Intelligence" by Jeff Hawkins in 2004. HTM was built on an understanding of the neocortex from a neuroscience perspective. HTM is similar to the neocortex structure and is a uniform hierarchy that works with invariant representation. Each representation can be separated into a cortical column. A column contains multiple neural cells inside. HTM connects its sensory and other cells by using dendrites and synapses. Proximal is used to receive input and feedforward. Distal is used for prediction. Each cell can learn and connect to others by

learning, called Hebbian Learning, wherein the connections will be strengthened if active together; otherwise, they decay. HTM constantly predicts its inputs and provides its outputs by using distal connections.

HTM provides a framework and fundamental mechanism for how the neocortex works by inspiring and simplifying this research and its efforts to create a cognition model inspired by the brain. However, this research proposes overall frameworks from the brain and additional improvement of fault tolerance, while HTM provides a theoretical framework.

C. Sparse Distributed Representations (SDRs)

SDRs [17] are information storage and transfer components in HTM. Information in SDR is kept in bits with "0" or "1" only. An SDR is a large vector of bits with only a tiny percentage active, which is the way the brain works with only a small amount of activity to reduce energy and inference. HTM can recognize both temporal and spatial. They use the Spatial Pooler mechanism and Temporal Pooling accordingly. This research uses the SDRs concept as the structure memory in HTM.

1) Definition: SDR is an n-dimensional vector of binary elements. SDR vector: $x = [b_0, b_1, \ldots, b_{n-1}]$, w_x is the number of elements in x that are bit "1". Overlap is the determination of the similarity between two vectors that is the number of bits that are 1 in the same location. For example:

X = [00000101010000000000001010100000000000000]
Y = [00010001010000010000001010000000000000000]

X and Y vectors have n = 40 and w = 6, *overlap* = 4 and sparsity is 15%, $s = \frac{w}{n}$ (6/40).

2) Matching: the possibility of the number of unique SDRs:

$$\binom{n}{w} = \frac{n!}{w!(n-w)!} \quad (1)$$

If n = 2048 and sparsity = 10% or w = 204, then the SDR space is 6.99×10^{286}. Thus, the probability of two random vectors being identical is as follows.

$$p(x = y) = 1/\binom{n}{w} \quad (2)$$

With n = 2048 and w = 204, the probability that two random vectors are identical is very close to zero.

3) Union: SDR can store a set of patterns in a single SDR using OR of all vectors. However, it increases the probability of false positives. With the number of union vector set, M, it becomes saturated with "1" bits, and almost random vectors will return a false positive match. The probability of a false positive can be written as:

$$p_{fp} = \left(1 - \left(1 - \frac{w}{n}\right)^M\right)^w \quad (3)$$

If n = 2048 and w = 204, storing M = 20 vectors, the chance of a false positive is 1 in 3.0×10^{11}.

IV. Conceptual

A. Overview of the New Cognition-Inspired Sequence Learning Memory

The brain processes input patterns that continue changing over time, called temporal patterns. A significant amount of data in real life also works with this kind of data using unique ordering and characteristics. These patterns are encoded into a sequence of invariant representations and proceed sequentially.

How do we learn sequentially? The brain processes a series of inputs one by one. At first, no sequence exists in the brain, and it comprises *short-term memory, long-term memory, and working memory*, which is the area of memory in our active focus. Once the first *representation* A comes in, the brain will create that representation in the working memory. The representation is kept in a neural unit. Once we remember step A, this representation will be activated. Afterwards, the second representation, B, arrives; it will also be created in the working memory as another representation or neural unit. However, the first and second representations will be connected automatically, called *auto-associate*, as shown in Fig. 1 (a). Once we try to remember something from one representation, such as step A, it will automatically remember the next step B. Thus, we can learn step A and can go to step B. The process is repeated for the next steps until the task is finished. We will also see that, at first, it is hard to remember step A, which the brain tries to find, but the brain will remember step B easily once we remember it.

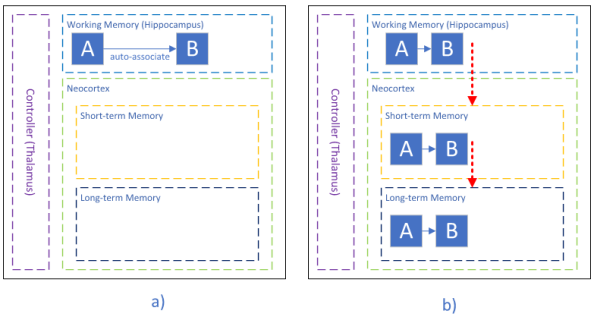

Figure 1. The concept of the learning sequence in the brain.

Even though we can remember it, we are still not sure about this sequence. We need a *filter* to see if this information is important enough to remember. Hence, we move it (A->B) to the short-term memory. Then, we can perform trial and error to see whether this sequence can achieve its goal or if we can see it often enough. Afterwards, we can be sure that this sequence is an important matter and promote it (A->B) to the long-term memory, as in Fig. 1(b). Hence, we can say we *learn* the sequence.

Therefore, each neural unit which is a representation in the brain can be activated one by one, sequential and in-reversible, representing what we focus on and think at that time. Once the first step is received and information is retained in the neural unit, it will automatically count to the next connected neural unit to receive the subsequent

input. Once the task is finished, these neural unit sequences will be moved to the long-short-term memory.

Accordingly, we keep sequence data that can contain many representations in a sequence table. We can define a set of representations, $R = \{r_1, r_2, r_3, ..., r_m\}$, A representation set X is a set of representations such that $X \subseteq R$. $|X|$ is the number of representations in a representation set X. *Count* represents how many times a sequence occurs, similar to the frequency in [18]. Each representation in a sequence is auto-associated automatically from left to right. The sequence in Table I represents sequences we keep in the brain that contain each sequence and its count. It is important to note that the input is received from sensory is encoded to invariant representations in a SDR vector form, found in Section IV-C.

Interaction with the world requires rapid processing. Further, we cannot learn every time or at each step that it takes us to think, process information, and make a decision. Because we already learned sequences for how to proceed, however, it remains in the long-term memory. Thus, the brain uses that information in long-term memory and *immediately predicts* the next step. Consciousness or *attention* arises only when the prediction fails. This mechanism is called the predictive mind or autopilot mode. Once a new task occurs, the brain perceives the first representation. Then, it will automatically determine what to do next from the long-term memory or what was learned previously. It is how we learn and use what we learn.

How is the situation handled if the prediction fails? Consciousness will arise to think and switch to learning mode. We can learn that this is a new thing called *online learning*. Sometimes, the input representation can contain a noise that makes the input sequence looks similar to a sequence in the long-term memory. How do we decide which sequence contains noise or is a new thing that should be learned? Hence, the difference between long-term and working memory sequences will be measured and determined. If the new input representation is not a prediction, we will switch to learning mode and wait to decide until the task is complete. If we decide it is noise, then we will correct it or just rehearse. However, it will be moved to the long-short-term memory if it is a new sequence.

Therefore, we need a mechanism to compare the sequence in the working memory and sequences in the long-term memory, which is a method proposed in this research. Each neural unit in the working memory is compared to each neural unit in a sequence in the long-term memory from start to end. If the number of different representations is low, it is noise—otherwise, it is a new sequence.

The next question is how we find related sequences and provide the next step when we perceive and perform a new task. For example, step A in the first input representation will be used as key "A" to find the long-term memory. Then, it will predict step B to be the next step from the long-term memory. Once step B is received, it will discover the next step from a new key, "A->B". In other words, the shortest key is used initially to find sequences from the long-term memory. Then, a set of possible sequences is returned as candidates. The key will continue growing in finding information from the long-term memory. The candidates will also continue to be removed from the previous candidates. This process will continue until the task is finished. Besides, a representation in the key can also be repeated, such as "A->B->A->B". Therefore, we must remember to pass steps A and B twice and keep them in our memory twice.

The concept is shown in Fig. 2. Each input is received and created as a key in the working memory to find the patterns inside the neocortex. The mechanism to describe how information is validated and found in the brain is introduced in the next Sections IV-B and IV-C.

B. Structure Memory

The structure of memory works hierarchically. One representation can contain and connect many representations in its lower layer. This concept is summarized in Fig. 2, which uses NLP as an example. In Layer 1, each presentation is a neural cell column representing one character, but a connection between representations is also a representation we remember as a sequence. In the more abstract levels shown in layer two or at the sentence level, representations and connections work the same as in Layer 1. This memory structure is based on the brain and inspired by Hawkins's approach, which uses SDR to provide a large vector of bits with only a small percentage.

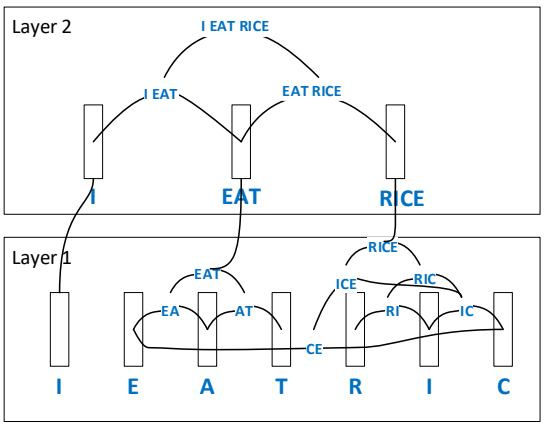

Figure 2. Forming connections for each representation and sequence pattern.

C. Fault Tolerance and Similarity

A crucial part of this approach is determining whether it is noise, what we know, or what we should learn by checking similarity. Instead of using weights to connect cells, we use bits or logical operations to process each representation.

This reduces complexity as well as processing time as it is easy to process, which is a problem for AI at present. Besides, it also supports using the modern memory structure that keeps information in bits. For example, not only are characters converted into bits but connections are also formed among them by using a hash function, as seen in Fig. 3, layer 1. Each representation or word can be compared by using logical operations. Furthermore, sentence comparison in Layer 2 or at the sentence level can

concatenate by using each representation of words and sentences from the lower layer.

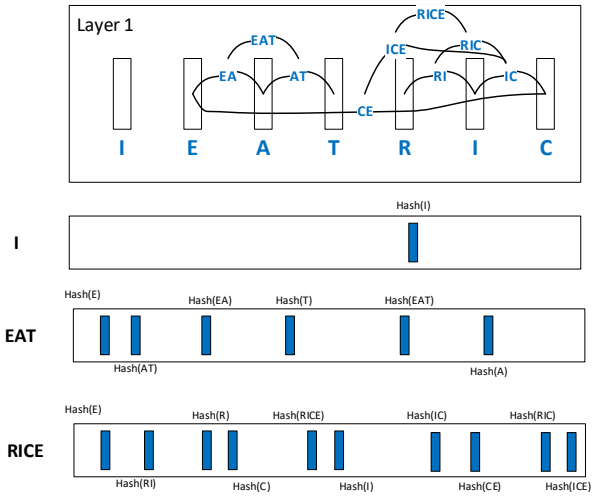

Figure 3. Converting representations into bits by using a hash function.

The authors do not specify the number of bits or representations. If too little, it cannot separate for each representation due to overlapping with other representations. If too much, on the other hand, existing memory could be used more efficiently. From the experiment, it can be around 1024–4096 bits for words in a dictionary.

For example, the size of SDR is 10 bits. It contains a zero vector [0 0 0 0 0 0 0 0 0 0]. The representation of "I" is hashing "I", hash ("I") %10=3, the output is [0 0 0 1 0 0 0 0 0 0]. The representation of "EAT" is hashing "E", "A", "T", "EA", "AT" and "EAT"; the output is [0 0 1 1 0 1 0 0 0 1]. Thus, if the misspelled word "EET" is received and its hashing vector is [0 0 1 0 0 1 0 1 0 1], similarity can be checked by AND or XOR operations to compare the similarity value, called the *diff*. For example, the *diff* is 3 (AND). However, comparing the similarity of "EET" is performed not only for the word "EAT", but all word vectors in the dictionary to find the highest number of *diff* (AND) or the lowest number (XOR). The operation is very fast as it operates at a bit level.

At the sentence level, it works in the same way but words are concatenated, such as "I EAT RICE". Three vectors are concatenated, and their size is 30, not 10.

TABLE I. SEQUENTIAL LEARNING TABLE

Sequence ID	Sequence	Count
1	{ B, U, T}	3
2	{B, A, L, L}	4
3	{E, A, T}	5
4	{B, A, T}	1

D. NLP Pattern Matching

According to Sections IV-B and IV-C, they provide understandings of how structure SDR vector is used and matched. This section is explained to summarize the concept and provide an NLP example as shown in Fig. 4.

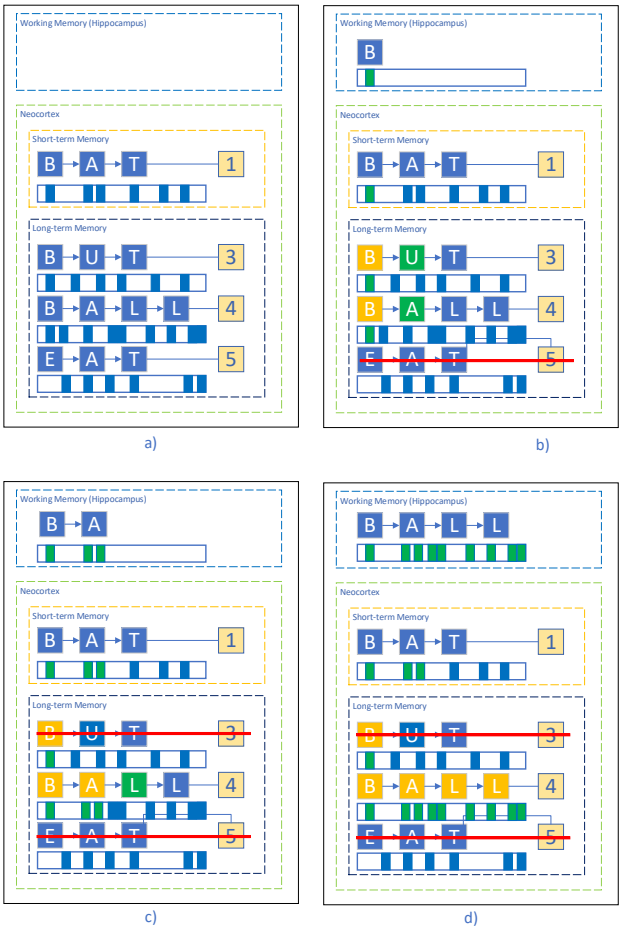

Figure 4. Sequential learning in a new cognition-inspired sequence learning memory.

At first, no data existed in the working memory. However, the neocortex comprised sequences that had their experiences as Table I. Each word sequence is encoded into a SDR vector. For example, the first word, "BUT" is kept in the memory by hashing, "B", "U", "T", "BU", "UT" and "BUT" to [0101010101010].

Once we perceive the first step, the brain will search any representations in the neocortex by using the first read element of hashing("B") as a key. Therefore, we can see that only the two first sequences in the long-term memory are SDR matched, except for the last sequence, which is removed since it does not match the representations, including the short-term memory that has not been remembered yet. Hence, the predicted steps are "U" and "A". "U" and "A" represent associative memory as it connects to "B" as well as two output lines, "U" and "A", which must be activated as they are predicted to be the next elements, meaning all output lines for a possible continuation of the sequence in the $i+1^{th}$ step will be activated. However, two cases are possible in case no continuation is found, as follows:

a) The representations have yet to be discovered and goes into learning mode.

b) An error element is read and then goes on reading the subsequent elements in an attempt to find out if possible solutions can be identified (fault tolerance).

We continue to perceive the second step, “A”; the brain will use both the first step, hashing(“B”), and then the second step, hashing(“A”) and hashing(“BA”), to search for a key. Thus, only the hashing pattern of “BALL” is matched with this key. It continues until the task is complete in the fourth step, “L”. Once it is completed, the working memory will be released.

E. Moving from the Short-Term to Long-Term Memory

The threshold of moving from the short-term memory to the long-term memory depends on the applications. In case of spelling check in offline learning, it can be set to 1 as training data should be correct words. However, in the hardware experiment in Section V is set to three (3) due to intuitive selection; if a human sees something three times, it should be able to be remembered. However, in case of online learning OCR spelling check, the question arises whether the system can learn from the OCR text. The assumption is the words from OCR can contain both correct and incorrect words and the OCR system should produce correct words more than incorrect words. For example, a word “eat” contains the correct word “eat” occurs five times and another word “ett” occurs two times. Thus, receiving the word “eat”, three times is enough. If the threshold set to three then it can use “eat” to learn correctly. Fig. 5 shows the number of occurring correct words and the average percentage of correct learning. As the result, setting n is 1000 can achieve the percent of the correct words at 74.45% because in some words the OCR system produces incorrect words more than correct words or never gives correct words. From the TREC-5 degrade5 data set, it is around 20% that the incorrect words are more than the correct words. Hence, the OCR application should learn from correct words or sentences that the threshold can be set to 1.

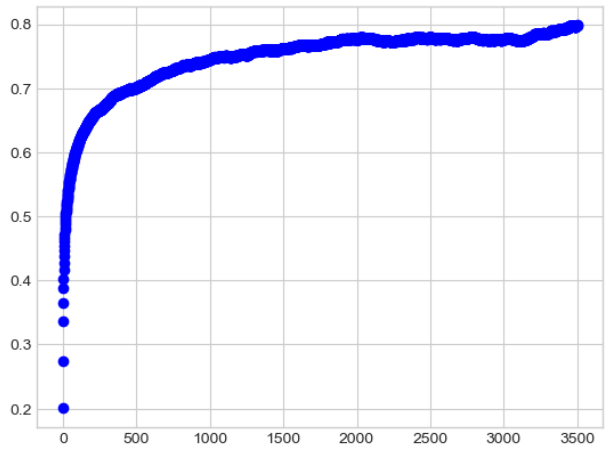

Figure 5. The number of occurring words and the average percentage of correct learning from TREC-5 Degrade 5%.

Nevertheless, in some application such as correction of search queries, the assumption is a user types the number of correct words more than incorrect words. Hence, it can learn from input of search queries. For example, qSpell [19], a data set from randomly sampled 11,134 queries from the publicly available AOL and 2009 Million Query Track. Fig. 6 shows the number of occurring correct words and the average percentage of correct prediction of search queries and found that if setting the threshold is 10, the correction learning can be 98.09% and never gives correct words are 109 words. However, this data set is low volume, by search engine characteristic with big data set correct words should be more than incorrect words and improve correction learning rate.

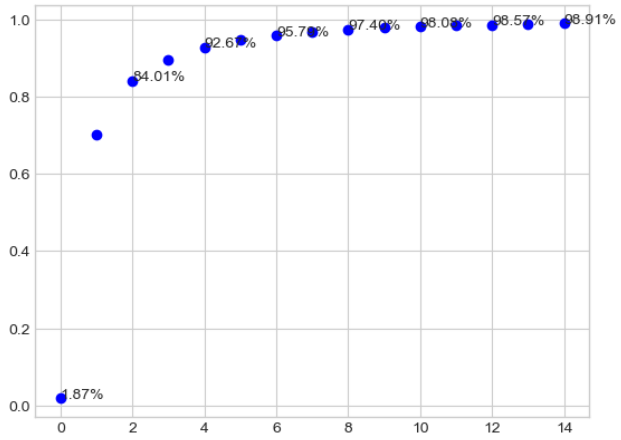

Figure 6. The number of occurring words and the average percentage of correct learning from qSpell search queries.

F. Big Data Set

Even though performing in bit operations is fast, it is also slower as its growth is linear. As an experiment, 7 million vectors can slow to 1 second. This research proposes two options to cope with and use in the evaluation. The first one uses parallel processing, as the current processor contains multiple cores and threads. The second option uses merge or union vectors. These options can work both offline and online.

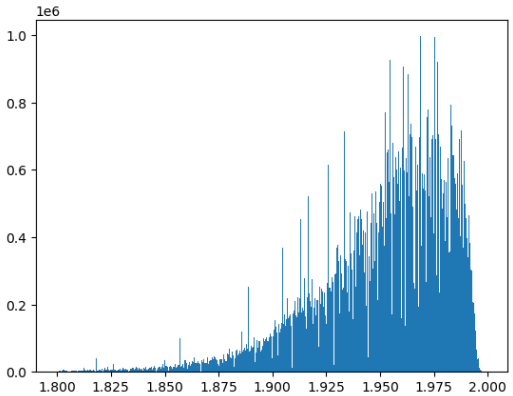

Figure 7. Plotting histogram of all distance vectors to find a suitable percentile.

Merging is finding the closet vector and aligning with it. The closest vector can be found by checking the similarity or distance as in Section III-C previously, after which two vectors will be merged into one to prevent a vector from containing too many presentations and not being unique. Two parameters are checked. The first parameter is a sparsity — how many percent one bit contains in a vector. The second parameter is how close a vector is to another vector. The second parameter is a distance threshold that

is measured by plotting a histogram of the distance of all vectors, as shown in Fig. 7, and selecting a percentile. For instance, for the Percentile at 90, the distance threshold is 1.975. After merging, sample words for a vector can be found, as in Table II.

TABLE II. MERGING VECTOR EXAMPLE

Vector	Merged words
1	(ACF), (ACF), (A-F), (ACK)
2	'bug', 'bug', 'bag', 'beg'
3	'baud', 'bad', 'bad' 'baud'
4	'scab', 'scab', ''cows', 'scow', 'scans'

V. EVALUATION

A. Hardware Simulation

Implementation is built on MATLAB Simulink software and takes advantage of the HDL module. The HDL module can design AI hardware for FPGA and ASICs suitable for this algorithm as it rapidly requests real-time interaction with the environment. The software is run on an ASUS TUF A15 laptop with an AMD Ryzen 75800H, 8 CPU cores, 16 threads, 32GB of RAM, and GPU RTX3060 6GB. Fig. 8 can be separated into five components as follows.

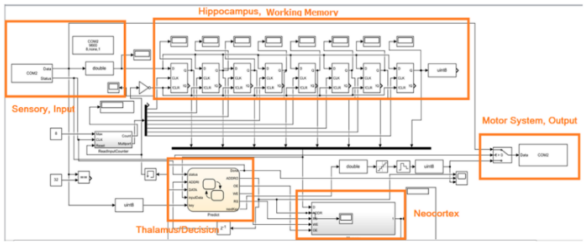

Figure 8. Overall hardware architecture of the cognition-inspired sequence learning memory on MATLAB Simulink.

1) *Sensory or input* connects to a comport and waits to receive input characters. Once the input is received, it will pass to the thalamus to proceed.

2) *Motor system or output* connects to a comport to provide output from the algorithm via the thalamus.

3) *Hippocampus, or working memory*, is used to keep information for a task until it is finished. Each input is processed via the thalamus or controller before being shifted to the working memory. The information is kept in a sequence.

4) *Neocortex or long/short-term memory* is where some information in the working memory will be moved to learn once the task is finished. If the sequence already exists, the word counter will be added by one. Otherwise, it will be added to the neocortex as a new sequence and the word counter is set to one. If the word count is higher than the threshold (3), it will be in the long-term memory. Otherwise, it will be in the short-term memory. The sequence can be forgotten as the information can be lost over time when not rehearsed. The neocortex implementation architecture is shown in Fig. 8. In this research, the threshold for moving information from short-term to long-term memory is set to three (3).

5) Thalamus or controller encodes and sends for both the encoded input and the input to the working memory each time an input is received to compare the representations of the working memory with all the sequences of the long-term memory. If the representations in the working memory is matched, it will use the next sequence in the long-term memory to predict and send it to the output. If not matched, however, it will send out "*", which represents "unknown" to the output. Once the task is completed, the controller will correct it automatically if the information is incorrect but close to a long-term memory pattern. This measurement is from checking the similarity of patterns and output as a score, called the diff. If the diff is not higher than the threshold, it is treated as an incorrect word, and the algorithm corrects it and has no learning. If the diff is higher than the threshold, however, the algorithm switches to learning mode and decides it is a new word, then moves to the neocortex to learn. The thalamus also is used to move and forget information from the working memory to the neocortex. A comparison among representations that checks similarity to find the score can be found in Fig. 9. The threshold diff can be calculated depending on the application or training data. For example, the threshold diff can be calculated from data set experiment such as degrade5 and used the average difference between ground words and input words of all training data set. The average is 45 but it can be adjusted according to an application.

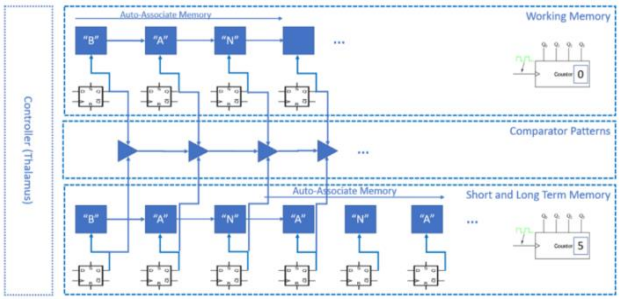

Figure 9. The concept for the architecture of the new cognition-inspired learning model hardware.

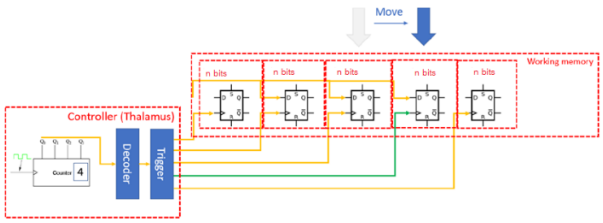

Figure 10. Working memory in the new cognition-inspired sequence learning memory.

This research sets each input to 2048 bits integer data type, which is more efficient than floating-point and reduces the bit-width to save energy and area, including increased throughput. Each input of data is fed into the algorithm one by one. Once one piece of data is received, the input data will be kept in a shift register or an address in the working memory. A counter in the controller will be

added by one for moving to the following address in the working memory. Each input data is kept in sequence until the task is finished or full. Addressing shall be increased at every step of the sequence reading. This mechanism can be found in Fig. 10.

One important note is that the input data is kept in sequence and auto-associated from left to right, which works like the brain that keeps patterns in sequences and reverse. Connecting for each input data or a shift register is controlled by the thalamus or controller. Once it needs to process information in a sequence, it will activate that information only. Besides, each sequence has a counter to support counting, called a data count, which represents the importance of information and remembers it via rehearsal.

If the data count is more than three, the sequence will be promoted to the long-term memory. Otherwise, it remains in the short-term memory. If the memory is full, the short-term memory with a lower data count will be removed first. Otherwise, the long-term memory with a lower data count will be deleted next.

Each received data input will be auto-associated with the previous input in the working memory to become sequence data. The algorithm will always predict the next step, which is the following pattern. If an unexpected prediction occurs, the brain will pay attention and decide to switch to learning mode or correct it when the task is completed. This representations in the working memory is compared with all sequence data in the long-term memory. For example, suppose all representations for the sequence in the working memory is matched with representations in the long-term memory. The found words will be used and predicted for the next pattern. If it is unpredictable, information will be kept in the working memory until the task is finished. Once complete, a correct decision as the following will be used.

Once the task is finished, the representations in the working memory will be compared with all the representations in the long-term memory (Neocortex). If they are exactly matched, the data count in the match sequence will be added by one. If the amount of different representations is not higher than the threshold, the data in the working memory will be corrected by a sequence of data in the long-term memory that has the lowest difference as we decide on a noise included. If the amount of different representations in the long-term memory is higher than the threshold, however, it will switch to the learning mode. The sequence in the working memory will be kept in the short-term memory as a new sequence and set the data count to one. However, if the sequence existed in the short-term memory, then the data count would be added by one.

For the experiment shown in Fig. 11, the Simulink MATLAB simulation is run on a laptop and connects the algorithm via a virtual comport to the experiment. There is no information in the algorithm, both the neocortex and the working memory at first. The first input is “c”. The algorithm shifts the input to the working memory and compares its representations with others in the long-term memory, after which the algorithm sends “*” or unknown to the output as no information existed. The next inputs are “a” and “t”, which are also unknown. We press <space bar> to end the task. The algorithm switches to the learning mode and learns the word “cat.” The sequence “cat” in the working memory is then moved to the short-term memory in the neocortex, and the word count is set to one. We repeat typing “cat” twice. This will promote the sequence “cat” in the neocortex to the long-term memory as the word count is changed to three (3). We type "c” again, and then the algorithm finds a match between the working and long-term memory to predict the next character, “a.” Then, typing “a” gives a prediction of “t.” We do the same with the word “abandon.” However, this time, we made a typo on the last word of the word abandon from “n” to “m” (abandom). The algorithm automatically corrects the pattern, and the correct output is "abandon,” as the score is less than the threshold.

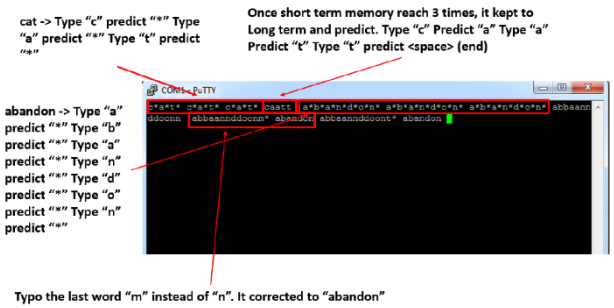

Figure 11. Experiment to connect to the algorithm via comport for both input and output.

B. *Performance Evaluation*

Two evaluation metrics are used to verify effectiveness and efficiency by using accuracy and average processing time. All measurements are performed on an ASUS TUF A15 laptop with an AMD Ryzen 7 5800H, 8 CPU cores, 16 threads, and 32GB of RAM, including GPU RTX3060 6GB. TREC-5 Confusion Track [2], the standard set, is used for evaluating an OCR spelling correction system. TREC-5 Confusion Track contains two corrupted versions of 55,600 documents that are created by applying OCR to page images. The first version is the scanned image page, estimated at approximately 5% of the error rate (degrade5). The second version is a down-sample of the page images, resulting in an estimated 20% (degrade20). After cleaning, it comprises 3,532,743 lines and 33,255,482 words, including 701,217 unique words.

Table III shows each method used to compare with the new method that contains the state-of-the-art, Neuspell (SC-LSTM plus ELMO at the input; Elmosclstm) and LSTM as well as standard methods, pyspellchecker, and Levenshtein Distance. Each method provides different types, as shown in the table. Only pyspellchecker is similar to this new method as it learns from dictionaries. In this experiment, pyspellchecker does not use the built-in dictionary, but rather the new vocabulary from the TREC-5 data set. In comparison, LSTM and SC-LSTM plus ELMO need to learn the error model for both correct and incorrect words. Therefore, a comparison between the error models and the methods learned from the correct words (the correct word methods) might not be appropriate

as the correct word methods will never know the correct words it never sees. Hence, the results from training data will be shown as well. Elmosclstm and LSTM use a GPU for training and running. LD, pyspellchecker, and the new cognition-inspired method use a CPU for training and running.

TABLE III. DIFFERENT TYPES FOR EACH METHOD

Method	Training Correct words	Training Incorrect Words	Learning
Levenshtein Distance	Yes	No	Offline
pyspellchecker	Yes	No	Offline
LSTM	Yes	Yes	Offline
SC-LSTM plus ELMO (at input)	Yes	Yes	Offline
The new cognition-inspired	Yes	No	Offline/ Online

In measurement, accuracy can be separated into three types. Accuracy means selecting the best score of words with only one, while AccuracyMax means selecting multiple words with the highest score if they have the same score. It can then sort the words with Levenshtein Distance. Finally, AccuracyRange means selecting some words (configured to 30 words) with n top scores, which can also be rechecked with Levenshtein Distance. For example, Input word = "W0rd", candidate words with a score composed of "Word" (4), "Ward" (4), "W0ad" (3), and "Wo9a" (1). "Word" (4) and "Ward" (4) have the same score. Accuracy selects only one, "Word", but AccuracyMax selects both "Word" and "Ward". AccuracyRange selects "Word", "Ward", and "W0ad" if the configuration is three words.

The evaluation is separated into small and big data sets using the degrade5 data set. The small data set that trains and tests sizes is 100,000 words and 12,961 unique words. The big data set uses training at 80% and testing at 20% of the entire degrade5 data set. The reason for evaluating both the small and big data sets is that some applications request only words in a dictionary that contains only a small amount of vocabulary, such as correcting search queries. Thus, there is no requirement for big data. In some cases, however, OCR words might contain not only a word in a dictionary but also an item number, page number, and specific patterns. Hence, the number of words can be too large.

1) Small data set

Table IV shows the performance of word level for each method. The new cognition-inspired method shows faster response over LD, pyspellchecker and LSTM by approximately 300 times, and Elmosclstm by 14 times because operations in bits are faster than other methods by creating 12,961 SDR vectors and use 2048-bit sizes. The accuracy of the cognition-inspired method shows an excellent result with the training data at 91.62%, meaning that the new method will get a good result if it learned the correct words. Otherwise, it will only get 75.02%, similar to methods. On the other hand, the accuracy of pretrained Elmosclstm is only 54.18% as it pretrained from random noise that does not contain this OCR data set characteristic. However, it got a better result after training of 82.18%.

TABLE IV. WORD-LEVEL PERFORMANCE ON A SMALL DATA SET

Method	Data	Accuracy	Accuracy max	Accuracy range	Time (s)/word
Levenshtein Distance (LD)	Test	71.06%	75.11%	79.49%	0.390
pyspellchecker	Train	70.75%			0.33
LSTM trained	Test	73.77%			0.328
LSTM trained	Test	71.94%			0.326
Elmosclstm pretrained	Test	54.18%			0.014
Elmosclstm trained	Train	82.18%			0.014
Elmosclstm trained	Test	75.29%			0.014
cognition-inspired 2048	Train	84.92%	91.06%	91.62%	0.001
cognition-inspired 2048	Test	71.07%	74.67%	75.02%	0.001

TABLE V. SENTENCE-LEVEL PERFORMANCE ON A SMALL DATA SET

Method	Data	Accuracy	Accuracy max	Accuracy range	Time (s)/word
Elmosclstm trained	Train	81.97%			0.023
Elmosclstm trained	Test	77.65%			0.025
cognition-inspired 2-gram	Train	93.33%	96.98%	99.31%	0.021
cognition-inspired 2-gram	Test	56.55%	64.03%	64.03%	0.022
cognition-inspired 3-gram	Train	97.24%	98.64%	99.08%	0.024
cognition-inspired 3-gram	Test	46.83%	49.42%	55.93%	0.024

While word level can fix non-word error types, it cannot fix real-word error types as it uses sentence level to handle them. This research compares Elmosclstm and the cognition-inspired method only for the sentence level as LD and pyspellchecker commonly work at the word level, and Elmosclstm is a similar approach to LSTM but newer and better for both accuracy and time. Table V shows the performance of sentence level for Elmosclstm and the cognition-inspired method. The cognition-inspired 2-gram is a concatenation between two words for every 2048 bits. Hence, the length of 2-gram is 4096 bits. Each word is concatenated as training sentences, and the number of vectors is 46,295 vectors. 3-gram is 2048 bits for each vector and it contains 64,432 vectors. As a result, cognition-inspired works well if it knows the correct words. The best accuracy is 99.31%. However, the accuracy of test data is only 64.03% for 2-gram and 55.93% for 3-gram because some words in testing data have yet to be learned. However, Elmosclstm provides a good result for testing data as it is already trained from pretraining. In summary, sentence level can give a better result than word level as it can correct the word from its surroundings.

2) Big data set

The big data set is separated into 80% for training data and 20% for testing data, as shown in Table VI.

TABLE VI. BIG DATA SET

	Line	All Words	Unique Words
Training 80%	2,780,211	26,604,373	701,217
Testing 20%	752,532	6651109	349,524

The big data set with word level in Table VII, LD is very slow at 13.14 s per word as it contains 701,217 unique words to search. Pyspellchecker provides the same result and works with the same concept as the small data set. LSTM is not tested for large data sets as it consumes both computing resources and time, while Elmosclstm can be represented better for both results and time. Elmosclstm pretrained does provide a poor result as it has yet to learn the OCR pattern. After training, the accuracy performance is better at 74.41% for testing data and 77.44% for training. However, the time performance of Elmosclstm still produces a good result as its architecture and parameters are the same. This result is close to its original paper, which works at around 79.8%.

TABLE VII. WORD-LEVEL PERFORMANCE ON A BIG DATA SET

Method	Data	Accuracy	Accuracy max	Accuracy Range	Time (s)/word
Levenshtein Distance (LD)	Train	80.83%	88.83%	94.83%	13.14
pyspellchecker	Train	76.02%			0.30
Elmosclstm pretrained	Test	58.88%			0.017
Elmosclstm trained	Train	77.44%			0.019
Elmosclstm trained	Test	74.41%			0.020
cognition-inspired 2048	Train	81.10%	87.38%	87.38%	0.05
cognition-inspired 2048	Test	79.35%	87.03%	87.03%	0.05

TABLE VIII. WORD-LEVEL PERFORMANCE ON A BIG DATA SET WITH MERGING

Bit size	Data	Merge Vector Size	Accuracy	Accuracy max	Accuracy Range	Time (s)/word
4096	Train	39,811	80.27%	87.48%	87.86%	0.01
4096	Test	39,811	76.45%	83.85%	84.31%	0.01
2048	Train	78,074	78.57%	81.83%	82.02%	0.02
2048	Test	78,074	79.02%	82.22%	82.49%	0.02

The cognition-inspired with 2048 bits and using 701,217 SDR vectors gives a better result than Elmosclstm with an accuracy of around 5%, the accuracy max, and a range of around 10%. However, the performance time is approximately twice as slow. The result is different from the small data set because the cognition-inspired already learned enough correct words.

Improvement by multiprocessing can contain overhead that also limits processing time. Thus, merging SDR is used first. The experiment set sparsity threads at 20% and distance threshold at 0.2 as plotting the histogram and set percentile at 90. The SDR vector size is also adjusted for merging with 2048 and 4096 bits. After merging, the vector sizes are reduced from 701,217 to 39,811 for 2048 bits and 78,074 for 4096 bits. In Table VIII, the accuracy for 4096-bit sizes produces similar results before merging, but the time is reduced significantly as the number of vectors is reduced by 20 times. However, 2048 bits give lower accuracy because extending the size of the vectors helps to reduce false positives.

TABLE IX. SENTENCE-LEVEL PERFORMANCE ON A BIG DATA SET

Method	Data	Accuracy	Accuracy max	Accuracy Range	Time (s)/word
Elmosclstm trained	Train	81.50%			0.015
Elmosclstm trained	Test	72.18%			0.017
cognition-inspired 4096	Train	84.97%	92.00%	96.73%	1.26
cognition-inspired 4096	Test	90.75%	90.83%	91.42%	1.95

Table IX shows the result of the sentence level in the big data set; Elmosclstm gives a better result as it uses sentence level to predict. Besides, its performance time is the same, which is a very good result. The cognition-inspired uses the word vectors from the previous experiment (701,217 SDR vectors) to construct the sentence vector and concatenate 2 words (2-ngram). After concatenation, the number of 2-ngram vectors is 3,972,879, converting a 4096-bit word vector to a 2048-bit word vector. Thus, 2-ngram that uses two-word vector concatenation still uses only 4096 bits. As a result, the accuracy is better than Elmosclstm for both training and testing data sets. Particularly, the accuracy range can provide 96.73% for training data and 91.42% for testing data.

Another experiment proves that cognition-inspired learning can provide high noise-tolerant attributes with no requirement for retraining the error model. Although the experiment uses a degrade20 data set, the cognition-inspired still uses the same training from degrade5 as it can learn from correct words only. Elmosclstm with retraining from degrade5 is used for evaluation. Elmosclstm shows low accuracy as it never retrains incorrect words from degrade20 to adjust the error model. However, the cognition-inspired still shows a good result of 90% and 94.50% for the accuracy ranges of both training and testing data, respectively, as shown in Table X.

TABLE X. SENTENCE-LEVEL PERFORMANCE ON A BIG DATA SET WITH A DEGRADE20 DATA SET

Method	Data	Accuracy	Accuracy max	Accuracy Range	Time (s)/word
Elmosclstm trained	Train	55.31%			0.016
Elmosclstm trained	Test	52.39%			0.017
cognition-inspired 4096	Train	69.17%	79.33%	90.00%	1.13
cognition-inspired 4096	Test	76.57%	81.67%	94.50%	1.25

The cognition-inspired shows a slow processing time of 1.26s for training data and 1.95 for testing data, however,

which is unacceptable for applications. To improve this, parallel processing is provided with threading that the method can apply easily. After submitting input to each thread, the result will be returned and combined in the main thread for sorting the score again. The parallel processing result can be found in Table XI. Currently, the number of threads is set to 10.

TABLE XI. SENTENCE-LEVEL PERFORMANCE ON A BIG DATA SET WITH PARALLEL PROCESSING

Method	Data	Accuracy	Accuracy max	Accuracy Range	Time (s)/word
cognition-inspired 4096	Train	85.10%	91.77%	93.24%	0.10
cognition-inspired 4096	Test	82.00%	83.12%	84.69%	0.13

In Table XI, the data is separated into ten parts after parallel processing. Thus, the processing time is reduced to about 0.1s. In some cases, however, the result might differ from one process because the method first calculates the scores from all SDR vectors and then ranges with LD. In other words, it finds word patterns similar to the input and then sorts them with edit distance. However, candidates will be selected from a part of all word patterns by separating information patterns into each thread. Hence, lower scores for checking SDR similarity can be in a candidate list and ultimately selected as it provides the lowest LD distance (but a different pattern). Nevertheless, the result is still better than Elmosclstm.

TABLE XII. SENTENCE-LEVEL PERFORMANCE ON A BIG DATA SET WITH MERGING

Method	Data	Accuracy	Accuracy max	Accuracy Range	Time (s)/word
cognition-inspired 4096	Train	75.80%	85.95%	89.62%	0.36
cognition-inspired 4096	Test	73.73%	77.39%	82.03%	0.357

Another approach is merging vectors as SDR attributes; two parameters are set, including sparsity and distance threshold. The vectors will not be merged by setting a low sparsity or high distance threshold. Setting high sparsity and distance threshold is low, vectors cannot be separated, and accuracy will be low. Merging from word vectors to SDR sentence vectors uses merged SDR vectors from the results in Table IX and concatenates them at the sentence level. This experiment set sparsity = 0.15 or 15%, and the distance threshold is 1.5. The number of vectors can be reduced from 3,972,879 to 908,410 or around 4.37 times. The result can be found in Table XII. The accuracy, accuracy max, and accuracy range give a good result over Elmosclstm, and even SDR vectors are merged and reduced. However, the time is still over Elmosclstm, but the cognition-inspired is still acceptable for most applications.

VI. CONTRIBUTIONS

This research provides a new cognition-inspired learning model inspired by the brain that provides benefits as follows:

1) The cognition-inspired model can work in offline and online learning modes, which is different from other methods that work in offline modes such as LSTM, Neuspell, LM, and LD or dictionary that cannot learn.

2) There is no requirement to learn from error words. Training error words can be an issue. Even if it can be generated randomly, each application has particular hidden patterns. Sometimes, it is almost impossible to produce correct and incorrect words to train. Learning only from correct words and sentences can be provided easily.

3) It can work in interactive spelling correction systems as it provides feedback and prediction.

4) It provides a new method for both small and big data sets. It works well for small data sets in terms of speed and accuracy if the correct words and sentences are trained. On the other hand, big data provides better accuracy with acceptable time.

5) The method can be implemented on commodity hardware.

6) The new approach provides high noise tolerance.

VII. CONCLUSION

This paper provides a new approach that was inspired by the brain and Hawkins. The approach proposes an overall framework that controls information (Thalamus) from working memory (hippocampus) to long-short term memory (Neocortex) that includes mechanisms, attention, and filtering. The information is kept in sequential and invariant representation structure as well as hierarchy level, similar to the neocortex in the brain. The paper also introduces a new method for determining the similarity between inputs and information in the brain that can tolerate noise by hashing it into bits and performing logical operations.

In a small data set and at the word level, the new method offers a faster response compared to other methods and still gets comparable accuracy. However, it performs with very low accuracy at the sentence level as it does not learn enough data. However, it provides excellent accuracy in training data. In big data sets, the accuracy is better compared with other methods, but the processing time is very slow. However, it can cope by using parallel processing or merging. The time is reduced significantly and it still provides a better result.

This research is open-source and available at https://github.com/thasayus/cognition-inspired.

CONFLICT OF INTEREST

The authors declare no conflicts of interest.

AUTHOR CONTRIBUTIONS

Thasayu Soisoonthorn conducted the research, created, tested the algorithm, and drafted papers. Herwig Unger

and Maleerat Maliyaem reviewed the manuscript and supervised the research. All authors had approved the final version.

REFERENCES

[1] M. Bruno and S. Mário, "Spelling correction for search engine queries," *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)*, vol. 3230, pp. 372–383, 2004.

[2] P. B. Kantor and E. M. Voorhees, "The TREC-5 confusion track: Comparing retrieval methods for scanned text," *Information Retrieval*, vol. 2, no. 2/3, pp. 165–176, 2000.

[3] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics — Doklady*, vol. 10, no. 8, pp. 707–711, 1965.

[4] F. J. Damerau, "A technique for computer detection and correction of spelling errors," *Commun. ACM*, vol. 7, pp. 171–176, 1964.

[5] H. Shang and T. H. Merrettal, "Tries for approximate string matching," *IEEE Transactions on Knowledge and Data Engineering*, vol. 8, no. 4, pp. 540–547, Aug. 1996.

[6] T. M. Miangah, "FarsiSpell: A spell-checking system for Persian using a large monolingual corpus," *Literary and Linguistic Computing*, vol. 29, no. 1, pp. 56–73, 2014.

[7] E. Ukkonen, "Algorithms for approximate string matching," *Inf. Control.*, vol. 64, no. 1–3, pp. 100–118, 1985.

[8] P. Norvig. *How to Write a Spelling Corrector*. [Online]. Available: https://norvig.com/spell-correct.html

[9] T. Barrus. (2018). Pyspellchecker-Pure python spell checker based on work by Peter Norvig. [Online]. Available: https://pypi.org/project/pyspellchecker

[10] A. C. Kinaci, "Spelling correction using recurrent neural networks and character level N-gram," in *Proc. 2018 International Conference on Artificial Intelligence and Data Processing (IDAP)*, Malatya, Turkey, pp. 1–4, 2018.

[11] S. Sooraj, K. Manjusha, M. Kumar, and K. Soman, "Deep learning based spell checker for Malayalam language," *Journal of Intelligent and Fuzzy Systems*, vol. 34, pp. 1427–1434, 2018.

[12] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in Neural Information Processing Systems*, vol. 4, 2014.

[13] S. M. Jayanthi, D. Pruthi, and G, Neubig, "NeuSpell: A neural spelling correction toolkit," in *Proc. the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 158–164, 2020.

[14] F. Amthor, *Neuroscience for Dummies*, John Wiley Sons Inc, 2016.

[15] X. Chen, W. Wang, and W. Li, "An overview of Hierarchical Temporal Memory: A new neocortex algorithm," in *Proc. 2012 International Conference on Modelling*, Identification and Control, Wuhan, China, pp. 1004–1010, 2012.

[16] H. Jeff and B. Sandra, *On Intelligence: How a New Understanding of the Brain will Lead to the Creation of Truly Intelligent Machines*, Macmillan, 2004.

[17] A. Subutai and H. Jeff, "Properties of sparse distributed representations and their application to hierarchical temporal memory," arXiv preprint, arXiv:1503.07469, 2015.

[18] M. J. Zaki, "SPADE: An efficient algorithm for mining frequent sequences," *Machine Learning*, vol. 42, pp. 31–60, 2001.

[19] Y. Ganjisaffar, A. Zilio, S. Javanmardi, *et al.* (2011). *qSpell: Spelling Correction of Web Search Queries Using Ranking Models and Iterative Correction.* [Online]. Available: https://www.ics.uci.edu/~chenli/pub/2011-Speller.pdf