Appendable memory system for artificial intelligence to acquire new knowledge after deployment

Kazunori D Yamada[1,2,*]

[1] *Unprecedented-scale Data Analytics Center, Tohoku University, Sendai, Japan*
[2] *Graduate School of Information Sciences, Tohoku University, Sendai, Japan*

Artificial intelligence (AI) that accumulates and presents knowledge to users has the potential to become a vital ally of humanity. Such systems should continually accumulate knowledge while operating without the need for relearning new data inputs. This study developed an appendable memory system for AI to acquire new knowledge even after deployment. Certain dialogue agents can acquire new knowledge after deployment and have the capability to make conversations based on past interactions stored in their memory, referred to as long-term memory, where the information is stored in natural language. However, existing dialogue agents have insufficient memory capacity to continually accumulate and share knowledge with users. This is because long-term memory must be processed by dialogue agents, and consequently, the format of the memory is limited to the modality of natural language, making it impossible to achieve a high compression efficiency. To continue accumulating knowledge post deployment, dialogue agents need memory that can store any amount of information in a finite-sized medium. This study aims to construct an AI system that can generate such memories. However, we demonstrate that such systems cannot be realized by the current machine learning methods, i.e., by merely repeating the input and output learning sequences with AI. Instead, this study proposes a method for storing any amount of information within a single, modality-unrestricted finite-sized vector and retrieving information from it by utilizing a neural network resembling a recurrent neural network and an encoder–decoder network, referred to as memorizer–recaller hereinafter. The proposed system can generate external memory, which is updated dynamically when AI acquires new knowledge; this is called appendable memory. Although this study yielded only preliminary results, it differs from traditional neural network learning methods and provides fundamental approaches for building an AI system that can store any amount of information within finite-sized data and can freely utilize them in the future.

KEYWORDS: learning with random values, neural networks, long-term memory, appendable memory, permanent memory

1 Introduction

A major challenge in the field of artificial intelligence (AI) is the construction of a strong AI or artificial general intelligence (AGI). The journey toward realizing these AIs is long and challenging. One challenge is the study of machine consciousness, i.e., endowment of AI with consciousness [1]. Another challenge lies in constructing entities that can physically exist in the real world. Furthermore, the ability to learn perpetually without needing a defined learning process like that found in the current machine learning (ML) field may also be necessary. Such an AI would continue to learn even after deployment. The development of this type of AI is a major challenge in the field of dialogue-agent research [2].

AI that can continue to learn post deployment must possess the following abilities.

(1) The ability to retain information that has been supplied thus far in the form of memory data.
(2) The ability to add the newly supplied information to existing memory data without compromising the already-memorized information.
(3) The ability to freely extract necessary information from memory data.

The abovementioned abilities are required by an AI to acquire new knowledge and utilize it according to the situation. Generative AI, particularly large language models (LLMs), have spurred immense growth in the field since 2022. The origin of this growth, ChatGPT, which is a dialogue agent developed by OpenAI, is capable of having a conversation based on its memory of a certain number (8,192 or 32,768 tokens) of past conversations [3]. Given that ChatGPT utilizes information from previous conversations, it would not be an exaggeration to consider ChatGPT an entity that continues to learn while operating as a dialogue agent. In addition, BlenderBot 2 developed by Meta, and its successor model BlenderBot 3, are dialogue agents that can summarize knowledge from past conversations, store it, and have conversations based on it [4]. BlenderBot 2 consists of a component called "long-term memory." This component consists of a summary of past conversations. The AI utilizes the information stored in the long-term memory to respond to user input. However, the information memorized is also finite, as in ChatGPT, owing to its naive implementation. Such LLMs are not designed to compress and retain past information in a different form from natural language, nor are they intended to decompress the compressed information and restore information from it. They are neural networks comprised of static parameters

*Corresponding author. E-mail: yamada@tohoku.ac.jp

trained to generate and process natural language. In other words, these dialogue agents do not have the capability to encode information into some other form of data or decode it. They are simply processing natural language. As long as memory information is stored in the form of natural language, the memory capacity will be limited.

In human conversations, people occasionally forget past conversations. If we seek the likeness of a human in AI, models with finite memory may partially meet this demand. AI with finite memory, such as ChatGPT, remembers knowledge acquired within a certain period entirely, and completely forgets knowledge learned prior to that period. However, people do not entirely forget the content of a conversation from a certain time ago and remember it after a certain time. In this sense, dialogue agents with finite memory may not resemble humans. Furthermore, we must decide if we need human-like imperfect features, such as memory capacity. As mentioned earlier, the major challenge of AI research is to construct a strong AI or AGI. The questions we have addressed in this study are: should all the abilities of strong AI or AGI resemble those of humans? Should AI possess imperfect memory? Strong AI is described as an entity with "spirit" by the study that first used this term [5]. This spirit is based on human nature. We believe that, for a strong AI to closely resemble humans, it should embody the fundamental essence of the human nature, encompassing its core attributes; however, its other abilities need not resemble those of humans. Instead, AI can even resemble machines in some other abilities: we express this feature as "machine-like." AI that retains information indefinitely, unlike human memory, and freely imparts knowledge without forgetting, would be a valuable companion to humanity. The insistence on AIs mimicking humans in every aspect is unnecessary. Indeed, many researchers hold this viewpoint. For instance, while animals do die, we do not wish the same for artificial entities such as Aibo, the dog-shaped robot developed by SONY. BlenderBot 2 has long-term memory but is not machine-like. Machine-like AI should be able to memorize any amount of information, and such a memory is a "permanent memory." In this study, we could not realize permenent memory but instead, we developed an "appendable memory" system towards permanent memory from using existing ML methods.

As mentioned earlier, we believe that machines should not need to forget. Current dialogue agents have finite memory because the modality of data they can handle is limited to natural language. In addition, the existing dialogue agents cannot retain previous information in a machine-readable memory format and utilize it for reference and information extraction. Past information, such as summaries of past conversations, must be in a form that AI can process as natural language. If memory information is stored as natural language, its capacity is inevitably finite. To overcome this limitation, we aim to create a system of AI that can remember past information in a memory format that can only be deciphered by machines, without being restricted by modality, and can extract information by referencing to it. However, such AI cannot be realized with networks such as dialogue agents, which only memorize the pattern between input and output values and return the output values to the input values. Such a network is an entity that has the function of returning output values to input values. However, the AI we desire to create is an entity that organizes newly acquired knowledge as "external memory," accumulates it there, and generates and utilizes the memory that can be updated. This leads us to an important question: is it possible to create an AI that can effectively utilize past information by current ML methods? Evidently, such a system cannot be realized using current ML methods. When constructing such a dialogue agent, we repeatedly train a model with pairs of questions and their corresponding response sentences. The same applies to LLMs, such as the generative pretrained transformer. For example, if a model that accepts a question and long-term memory as input data and returns a response sentence is trained repeatedly using conversation pair data, the model will stop focusing on the memory as learning progresses. This is because the trained model is optimized for the training data and thus can return an appropriate response to a question without referring to the information in the memory. Exploiting information that can only exist in the external memory, such as long-term memory, to generate a response from AI may be impossible within the current ML framework. In other words, the extant dialogue agents cannot "reference" memory information other than natural language, nor can they "extract" information from it. Dialogue agents are natural language processors; thus, if a device could retain past information indefinitely in natural language, the dialogue agent would be capable of considering an infinite amount of historical information when responding to the user.

To answer the aforementioned question, we conducted an experiment with encoding and decoding operations. We used two models, "memorizer" and "recaller." The encoding operation is referred to as the "memorization" operation, and the decoding operation as the "recall" operation. In the experiment, during the memorization process, we generate a "memory vector" that encapsulates various pieces of information. The memory vector in this experiment corresponds to the long-term memory, permanent memory, or appendable memory. Next, we use this memory vector and a piece of data that triggers the system to restore information from the memory vector as input values for the recaller. We then test whether the system can extract information from the memory vector. Through this experiment, we demonstrate that, while current ML methods can extract patterns inherent in data, they cannot teach AI how to remember or recall information.

In this study, we conducted an experiment to demonstrate the limitations of the existing learning methods and proposed a learning method designed to avoid overfitting the model to patterns inherent in the input dataset. In addition, we designed a system that can encode multiple pieces of information independently into a memory vector and restore the information freely. Neural networks trained using conventional learning methods store the memory information necessary to return appropriate output values for the input values within its parameters, and we consider such a neural network consists of "static memory." In contrast, the proposed system can generate "dynamic memory," which is referred to as appendable memory. It is a type of memory that can change dynamically without learning a neural network or changing its parameters. From the next section, we will first discuss the limitations of current ML methods, then propose a ML

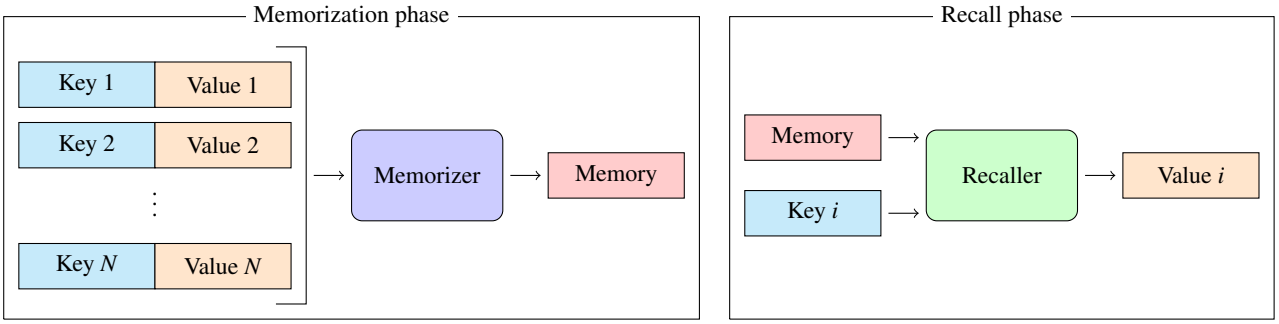

Fig. 1: Details of computation in the memorization and recall phases. In the memorization phase shown in the left panel, N pairs of key and value combinations are used as inputs to the memorizer in no particular order, which results in an output value, which is the memory vector. In the recall phase shown in the right panel, the memory vector and a key, which are parts of the input information from the memorization process, are used as inputs to the recaller to output the value vector corresponding to the key vector. In the diagram, rectangles represent data, and rounded rectangles represent operations.

method using random numbers to realize appendable memory, and finally describe applications that utilize appendable memory.

2 Experiments and Discussion

2.1 Network Architecture

The aim of this study is to train the memorizer–recaller network such that the memorizer can store N pairs of key–value vectors into a memory vector, while the recaller outputs the corresponding value from the memory vector and one key vector, as shown in Figure 1. In other words, we need to restore complete information from the key using the memory vector. Using a dialogue agent as an example, this operation can be explained as shown in Figure 2. The combination of key and value corresponds to a statement by a particular user, and the memory vector represents the accumulation of these statements stored in the AI system. The key represents some piece of information, and the value is an explanation of this information. The role of a recaller is to recall the value corresponding to a key based on its memory. If an AI can have this function, it would be able to accumulate knowledge from users or even web searches and grow without changing or updating the parameters of the neural network after deployment.

The operations performed by the memorizer–recaller network can be achieved with a key–value store in cases where the key and value are simple combinations and the memory capacity is unlimited. However, in this study, the size of the memory vector is fixed and that of data to be memorized is arbitrary N. Moreover, if we assume natural language as the input value, the key and value are not always in this order: instead of the tandem key–value structure, the value may also be located in the middle of the key vector. These types of data cannot be handled using a simple key–value store. Furthermore, the desired operation with this memorizer can be realized by a neural network equipped with an attention mechanism, that is, an attention neural network (ANN) [6]. By considering past contextual information, ANNs can receive sequential values as input and output values. In this study, we performed a similar operation using the memorizer. However, one key distinction between the memorizer and conventional ANNs is that ANNs cannot append information to a memory vector, although it can overwrite a memory vector.

We used two neural networks and presented the relevant formulas of the models. The left arrow ($\leftarrow$) is a symbol that assigns the value on its right to the variable on its left. The first model, that is, the memorizer, is represented by the following formula:

$$u_t \leftarrow f(k_t, v_t), \tag{2.1}$$

$$p \leftarrow \sigma(w_1 u_t + b_1), \tag{2.2}$$

$$q \leftarrow \sigma(w_2 m_{t-1} + b_2), \tag{2.3}$$

$$m_t \leftarrow \sigma(w_3(p+q) + b_3), \tag{2.4}$$

where w denotes the weight parameter, b is the bias parameter, and σ is the activation function. We used LeakyReLU [7] as the activation function. In addition, f is a function that concatenates two vectors, t is an integer from 1 to N, k_t is the t-th key, v_t is the t-th value, and m_t is the t-th memory vector. In the memorizer, the $(t-1)$-th memory vector is used to generate the t-th memory vector, and the memorizer has a recurrent loop in its structure; therefore, the memorizer is

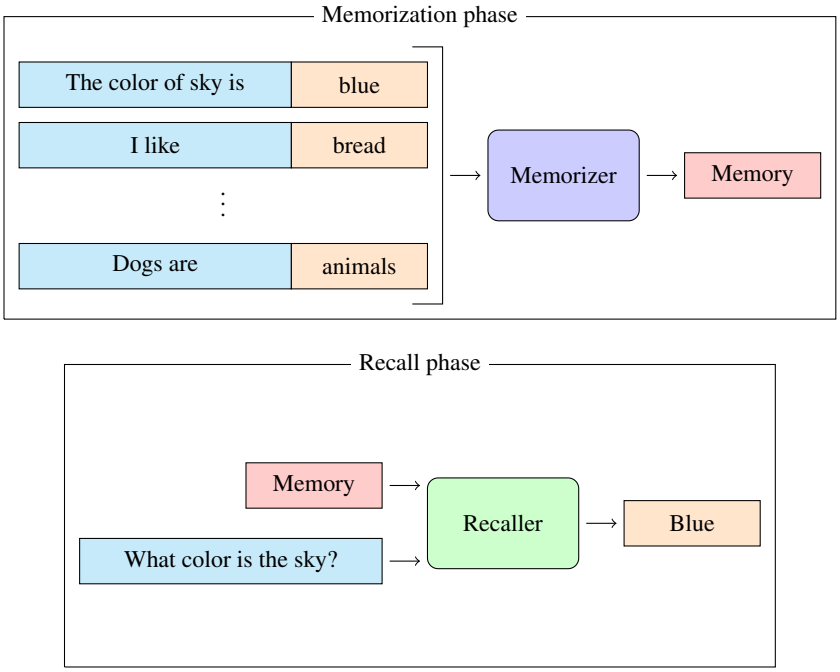

Fig. 2: Example of an application of the memorizer–recaller network as a dialogue agent. In the diagram, rectangles represent data, and rounded rectangles represent operations.

a type of recurrent neural network (RNN). The variable p in the final formula is derived from the input data, whereas q is derived from the $(t-1)$-th memory vector. The memory vector includes both previous memory information and new input information. Because calculating the elementwise summation of p and q is essential, these vectors must have identical dimensions. We performed computations for all N data points during the memorization phase to obtain m_N. The second model, that is, the recaller, is represented by the following formula:

$$r \leftarrow \sigma(w_4 k_i + b_4), \tag{2.5}$$

$$s \leftarrow \sigma(w_5 m_N + b_5), \tag{2.6}$$

$$\hat{v}_i \leftarrow f(r,s), \tag{2.7}$$

$$\hat{v}_i \leftarrow \sigma(w_6 \hat{v}_i + b_6), \tag{2.8}$$

$$\hat{v}_i \leftarrow \phi(w_7 \hat{v}_i + b_7), \tag{2.9}$$

where ϕ is the activation function, and because the recaller is used to classify and predict values from 0 to 9, we used the softmax function. In addition, k_i is any one of the N keys used as the input to the memorizer. In the recall phase, it attempts to output $\hat{v}_i$ corresponding to k_i.

Figure 3 illustrates a neural network. In the figure, the structure on the left is the memorizer, and the one on the right is the recaller. The memorizer accepts pairs of key and value vectors as inputs and outputs a memory vector. Repeating this calculation for all N input values ultimately generates a memory vector that is expected to contain all input information. The recaller accepts a single key and the memory vector generated by the memorizer as input and outputs the value corresponding to the key. The objective of using the memorizer–recaller network is to identify a trained neural network that can realize the storage and retrieval of multiple pieces of information, as shown in Figure 1, without requiring relearning. In this study, we have not discussed the performance enhancement of the system; therefore, we did not include any luxury mechanisms, which are generally used to improve the performance of neural networks. We designed the system with a simple combination of layers, as illustrated in the figure. The source codes used to generate all the networks in this study are available in a GitHub repository git@github.com:yamada-kd/Memorizer-recaller.git.

2.2 Learning Process of the Models

2.2.1 Learning in a Standard Manner

The next step in the learning process is the training of the model. First, we explain the hyperparameters used for training. The size of the intermediate layer of both the memorizer and the recaller was set to 256. Similarly, the dimension of the memory vector, which is the output of the memorizer, was set to 256. Because the recaller is a predictor that

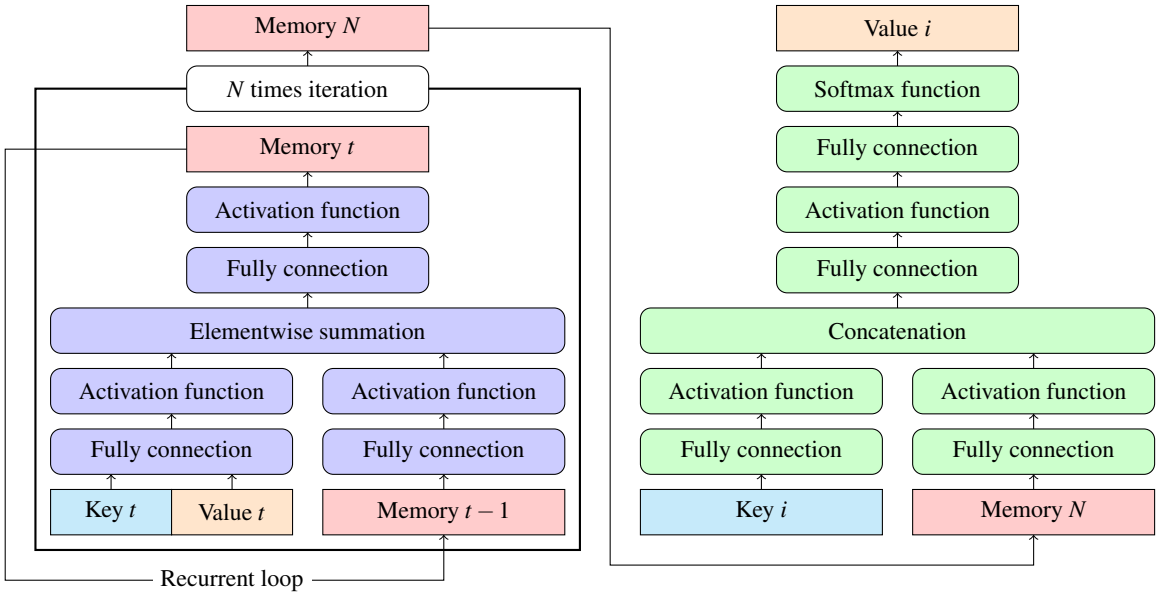

Fig. 3: Illustration of the memorizer–recaller network. The network on the left is the memorizer, which accepts N pairs of key and value vectors as input, and outputs the memory vector. The recaller on the right accepts a single key and the memory vector as input and outputs the corresponding value. The rectangles represent data and rounded rectangles represent operations.

outputs integers from 0 to 9, the dimension of its output vector was set to 10. In addition, the dimension of the key vector was set to 16. Each element of the key vector is a floating-point number arbitrarily generated from a continuous uniform distribution, followed by parameters with a minimum value of 0 and maximum value of 9. The value vector is one-dimensional, and its only element is an integer randomly generated from a discrete uniform distribution with a minimum value of 0 and a maximum value of 9. The volume of data to be input to the memorizer is N; N is variable for the experiment. For training, 1,024 combinations of these N input data were generated and used in the batch learning method. During the learning process of the memorizer, the value of t was varied from 1 to N, the memory vector m_t was calculated, and finally, m_N was computed. For m_0, random numbers generated from a uniform distribution following the parameters with a minimum value of -1 and a maximum of value 1 were used. Because the softmax function was used in the output layer, the cross-entropy error function was used as the cost function. The parameters were updated when the error between the value and teacher data was calculated after inputting m_N and each of the N keys to the recaller to output the predicted value. Therefore, the cost function L was computed as follows:

$$m_t = M(k_t, v_t, m_{t-1}), \tag{2.10}$$

$$\hat{v}_i = R(k_i, m_N), \tag{2.11}$$

$$L(\mathbf{w}, \mathbf{b}) = \frac{1}{N}\sum_{i=1}^{N} l(\hat{v}_i, v_i), \tag{2.12}$$

where M represents the memorizer, R represents the recaller, l denotes the cross-entropy error function, $\mathbf{w}$ signifies all weight parameters of the memorizer–recaller network, and $\mathbf{b}$ denotes all bias parameters of the memorizer–recaller network. The parameter optimization method used was Adam [8].

The results of the training are listed in Table 1. During training, the training and validation datasets were randomly generated. In other words, there is no discernible rule or pattern that the ML model can discover in each piece of data. Therefore, to derive the correct answer in the validation phase, the recaller must refer to the memory vector generated during the memorization phase. The training was terminated when the accuracy in the training phase reached 0.8. Table 1 displays the number of epochs required and the accuracy of the validation dataset. As a result of the training, the accuracy in the training phase was 0.8; however, the accuracy in the validation phase was still low; thus, we could not build a sufficiently good model.

It was difficult to construct a robust model during the training phase because the model overfitted the training dataset and adapted excessively to it. In other words, this model learned the characteristics of the training dataset consisting of random elements as well. Therefore, it could not adapt to the validation dataset, which had no correlation and did not share any characteristics with the training dataset. Although the recaller should ideally refer to the memory vector,

N	Epoch	Training accuracy	Validation accuracy
2	550	0.802	0.233
3	650	0.802	0.168
4	820	0.800	0.146
5	1000	0.801	0.139
6	1100	0.801	0.120
7	1510	0.800	0.118
8	1470	0.801	0.113
9	1720	0.800	0.111

Table 1: Results of training with various number of input data N. Training was terminated when the accuracy for the training dataset reached 0.8. Epochs represent the learning time required for the accuracy of the models to reach this point.

which is a condensed medium of the information input to the memorizer, in order to derive answers, it merely attempts to output a response based on the pattern of the randomly generated vector. In other words, in a learning task, the model predicts the output values corresponding to the input values. However, incorporating the function of referencing to the memory vector is challenging because the model can generate output values corresponding to the input values with more learning iterations. We believe that the parameters of neural networks trained by the standard learning method are a type of memory that memorizes the relationship between the key and the value. As mentioned earlier, we refer to this memory as static memory. Once a system holds perfect static memory overfitted to the training dataset, it need not refer to an external memory such as the memory vector. This is because AI constructed with this conventional ML method is the one that can return appropriate output values for input values, and it does not acquire the function to store knowledge in the memory vector or extract information from it.

2.2.2 Learning with Random Values

The objective behind proposing a memorizer–recaller network is to identify a "way" of organizing input values, memorizing them to the memory vector, and a way of searching for necessary information from memory and recalling it. In the conventional ML training methods described in the previous section, a model learns the characteristics of the data and attempts to understand the patterns inherent to them. An LLM that has extensively learned a text can be considered to have discovered some rules for language generation. In contrast, this study aims to construct a system that can memorize and recall information regardless of the scale of the learning data or the domain from which they originated. The proposed model is not a system to search for patterns inherent to the data, but a type of mnemonic system to create a model and learn how to memorize information.

In this study, we removed the patterns from the learning data to prevent the model from recognizing patterns inherent in the data and enable it to learn how to behave. Specifically, we probabilized k_i, which is the input value to the memorizer and recaller, and v_i, which is the input value to the memorizer and teacher data for the recaller. During the learning process, each of the 16 elements of k_i were randomly generated from a continuous uniform distribution, with parameters ranging from 0 to 9 for each epoch. Only the element of the value vector was generated from a discrete uniform distribution with a minimum value of 0 and a maximum value of 9. Although we used randomly generated data in the learning process, the aim was to ensure that random data do not have any patterns. The objective of the study was to train a memorizer–recaller network that could acquire the skills of remembering and retrieving information from memory, rather than simply learning the underlying patterns of data.

The results of learning using this method are listed in Table 2. The learning process ended when the validation accuracy reached 0.8. In general, the accuracy of the validation dataset is lower than that of the training dataset. If this is not the case, abnormal learning settings may be suspected. However, because both the training dataset and validation dataset in this learning method are random values, if the learning progresses normally in the training dataset, the accuracy in the validation dataset improves simultaneously, and this behavior is normal.

We conducted training while varying the volume of data N to be memorized from 2 to 9. Until N reached 8, the accuracy reached 0.8 for both the training and validation datasets. In other words, the trained memorizer–recaller was able to derive the correct answer in a validation dataset by learning from a training dataset where both the training and validation dataset were randomly generated and thus had no patterns. However, when N was 9, despite training the model for 500000 epochs, the accuracy did not reach 0.8, indicating that the learning process did not proceed successfully. In other words, the memorizer–recaller constructed under these learning conditions could memorize at most 8 pieces of information.

Next, to demonstrate the memorizer–recaller 's performance, we sequentially input 8 sets of key and value data into the memorizer to generate a memory vector, as illustrated in the upper panel of Figure 4. Subsequently, we conducted a

N	Epoch	Training accuracy	Validation accuracy
2	5580	0.800	0.800
3	9200	0.800	0.800
4	13920	0.800	0.800
5	19800	0.800	0.800
6	30600	0.800	0.800
7	53140	0.800	0.800
8	95280	0.800	0.800
9	500000	0.259	0.259

Table 2: Results of training with input data N where the training dataset is randomly generated for each epoch. Training was terminated when the accuracy for the validation dataset reached 0.8. Epochs represent the learning time required for the accuracy of the models to reach this point.

The input data to the memorizer

(1) 8.75, 7.90, 4.59, 0.500, 4.06, 0.180, 3.98, 8.82, 3.24, 4.33, 6.20, 7.92, 8.26, 1.95, 5.09, 7.79, **7**

(2) 3.46, 2.68, 0.510, 2.45, 4.30, 7.31, 4.32, 3.54, 7.52, 3.04, 5.83, 3.31, 8.61, 1.26, 7.83, 4.26, **5**

(3) 3.48, 8.12, 4.05, 5.52, 8.12, 0.890, 8.73, 5.88, 1.54, 3.22, 6.76, 5.47, 2.93, 0.350, 5.71, 8.63, **3**

(4) 5.88, 5.72, 8.96, 5.24, 3.73, 4.27, 5.61, 3.04, 6.07, 2.85, 7.01, 8.55, 5.96, 0.120, 5.61, 6.06, **3**

(5) 2.82, 8.69, 5.30, 5.94, 4.80, 2.07, 3.55, 5.57, 4.27, 4.23, 6.44, 2.59, 3.45, 6.74, 7.91, 0.930, **5**

(6) 7.21, 4.68, 6.11, 6.49, 5.24, 4.84, 6.83, 0.950, 4.26, 1.68, 6.63, 1.95, 1.22, 2.92, 1.35, 2.00, **0**

(7) 4.58, 8.25, 8.29, 0.750, 2.50, 0.0800, 7.58, 5.82, 7.57, 2.38, 3.58, 4.98, 1.48, 3.33, 1.32, 5.13, **9**

(8) 6.33, 2.60, 3.90, 6.80, 3.56, 8.06, 5.75, 8.02, 6.12, 4.04, 8.81, 1.05, 6.90, 3.71, 6.08, 2.25, **3**

The input data to the recaller and the output data generated by the recaller

- 3.46, 2.68, 0.510, 2.45, 4.30, 7.31, 4.32, 3.54, 7.52, 3.04, 5.83, 3.31, 8.61, 1.26, 7.83, 4.26 : 5 (**5**)
- 7.21, 4.68, 6.11, 6.49, 5.24, 4.84, 6.83, 0.950, 4.26, 1.68, 6.63, 1.95, 1.22, 2.92, 1.35, 2.00 : 0 (**0**)
- 3.48, 8.12, 4.05, 5.52, 8.12, 0.890, 8.73, 5.88, 1.54, 3.22, 6.76, 5.47, 2.93, 0.350, 5.71, 8.63 : 3 (**3**)
- 5.88, 5.72, 8.96, 5.24, 3.73, 4.27, 5.61, 3.04, 6.07, 2.85, 7.01, 8.55, 5.96, 0.120, 5.61, 6.06 : 3 (**3**)
- 8.75, 7.90, 4.59, 0.500, 4.06, 0.180, 3.98, 8.82, 3.24, 4.33, 6.20, 7.92, 8.26, 1.95, 5.09, 7.79 : 7 (**7**)
- 4.58, 8.25, 8.29, 0.750, 2.50, 0.0800, 7.58, 5.82, 7.57, 2.38, 3.58, 4.98, 1.48, 3.33, 1.32, 5.13 : 9 (**9**)
- 6.33, 2.60, 3.90, 6.80, 3.56, 8.06, 5.75, 8.02, 6.12, 4.04, 8.81, 1.05, 6.90, 3.71, 6.08, 2.25 : 3 (**3**)
- 2.82, 8.69, 5.30, 5.94, 4.80, 2.07, 3.55, 5.57, 4.27, 4.23, 6.44, 2.59, 3.45, 6.74, 7.91, 0.930 : 5 (**5**)

Fig. 4: Example of a test conducted using the trained memorizer–recaller network. The input data to the memorizer are shown in the upper panel. The data numbered 1 through 8 were input into the memorizer. The input data to the recaller and output data generated by the recaller are displayed in the lower panel. These input data were randomly input into the recaller to obtain the predicted values. The values in parentheses associated with each input and output data in the lower panel represent the correct values.

test using all keys to determine whether the values could be retrieved using the memory vector and recaller. The integers in bold represent the values. The results of this test, in which the output values of the recaller next to the colon were recorded, are depicted in the lower panel of Figure 4. In this case, the recaller retrieved the correct values for all keys. This test was repeated 1,024 times, and the mean accuracy was 0.916. This indicates that the memorizer–recaller network exhibited good performance even in the tests. Moreover, it could memorize multiple pieces of information in a single memory vector and retrieve relevant information.

Next, we examined the changes in mean accuracy when the memorizer–recaller constructed for $N = 8$ processed different volumes of data. In this experiment, the mean accuracy was computed by repeating the test 1,024 times. The

The number of input data	Accuracy
2	0.967
3	0.938
4	0.941
5	0.931
6	0.926
7	0.912
8	0.916
16	0.510
32	0.301
64	0.204
128	0.151
256	0.126

Table 3: Accuracy of the memorizer–recaller, which was trained by setting N to 8 when processing different numbers of input values.

results, listed in Table 3, demonstrate an accuracy exceeding 0.9 for the training dataset comprising 8 data points. However, the performance significantly drops when the amount of data processed increases beyond that. This is a 10-class classification problem; therefore, the accuracy for random answers was 0.1. When the volume of data to be memorized was set to 256, the accuracy was almost the same as that when answering randomly.

The memorizer–recaller network had distinctive characteristics in the way deriving the correct answers. The recaller generated the correct output for the last 8 pieces of information memorized by the memorizer; however, it was unable to recall the data provided earlier. For instance, when the input data consisted of 16 items, the memorizer–recaller exhibited an accuracy of 0.510, which largely exceeded the accuracy of 0.1 when making random predictions. However, this was because the memorizer–recaller could derive the correct answer with an accuracy of 0.510 uniformly for the data to be memorized. The value of 0.510 was calculated as the average with an accuracy of approximately 0.9 in deriving the correct answer for the last 8 pieces of data input to the memorizer and approximately 0.1 for deriving the correct answer for the first 8 pieces of data.

Using the developed learning method, we could construct a system that could remember multiple pieces of information in a single vector, despite being a trained model, and retrieve them for a small number of items, i.e., 8 items. The memorizer–recaller network is similar to an encoder–decoder network, such as a transformer, in terms of the neural network structure, and the computational processing in the memorizer is an extension of ANNs or RNNs. However, by effectively utilizing random data in the learning process and not allowing the same input data to be learned twice, we ensured the AI remembered the procedure to solve problems instead of learning patterns embedded in the data. Consequently, the network can continue learning after the learning process, which distinguishes it from existing AI. However, the memory capacity is extremely small; thus, the memory vectors generated by the memorizer are far from being called permanent memory. To improve this, further investigation is needed, not mere ad hoc measures such as increasing the size of the network parameters.

A feature of the recaller that differentiates it from previous neural network models is that it restores information from memory vectors that possess properties distinct from those of the input values. As discussed in the previous section, we demonstrated the possibility that neural network models may not be able to utilize external memory effectively because of their adaptation to learning datasets. However, AI models, such as ChatGPT and BlenderBot 2, can refer to previously learned memories and information. As previously mentioned, this is fundamentally different from what a recaller does. ChatGPT and BlenderBot 2 combine prior information and queries and employ them as inputs for a static model to generate responses. In the end, LLMs are nothing more than AI capable of processing natural language. No restoration of any type of information occurs from memory vectors in a format distinct from natural languages. Furthermore, the memorizer differs from the previous neural network models in the ability to append new information to memory vectors, which is a key feature of memorizer–recaller networks. A memorizer–recaller network possesses a dynamic external memory in the form of memory vectors. Considering that ChatGPT and BlenderBot 2 incorporate past input and output values as well as their summaries, along with current input values, these past pieces of information are regarded as external memory. In the case of ChatGPT, this external memory is a batch of sentences with a certain number of tokens, whereas BlenderBot 2 achieves this through several summary sentences. Conversely, the external memory of the memorizer is a fixed-sized vector appendable to new information. We intend to explore the future potential of appendable memory vector. Although the volume of information that can be remembered depends on the size of the memory vector and precision of floating-point numbers, we aim to improve this by making the model retain additional information in the future.

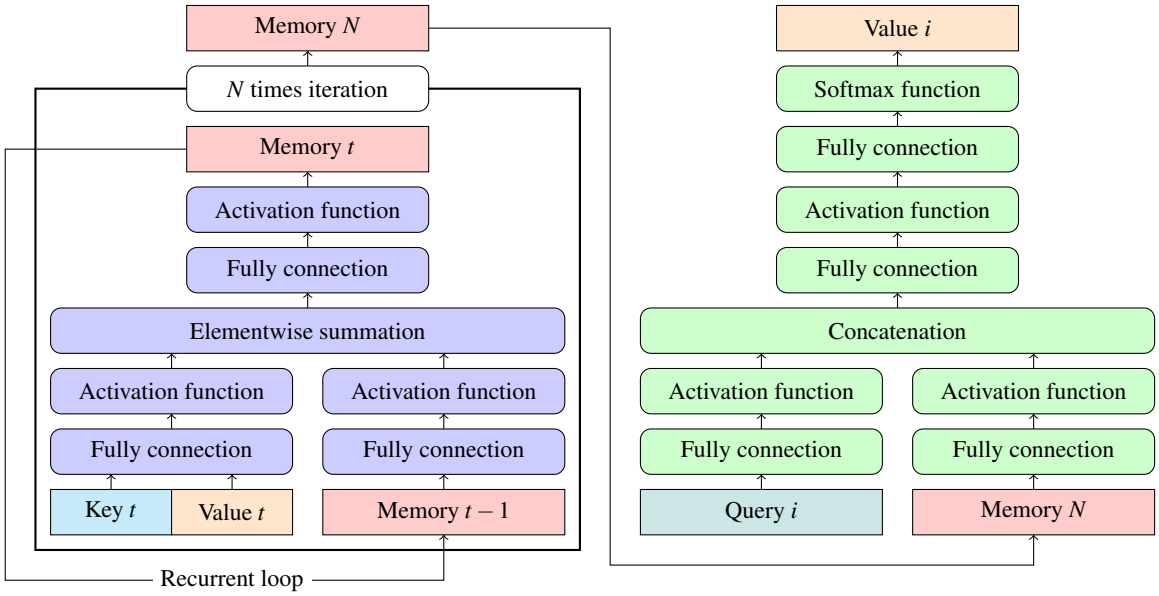

Fig. 5: Illustration of the network when generating a sorting algorithm with the memorizer–recaller. The network structure is the same as the previously mentioned memorizer–recaller network; however, a part of the input data of the recaller was changed to a query. In the diagram, rectangles represent data and rounded rectangles represent operations.

2.3 Development of Sorting Algorithm

Next, we explored whether a sorting algorithm can be generated using the memorizer–recaller network. If a model can remember the input information, it can sort that information. As the memorizer–recaller system can memorize the input information in a memory vector, we conducted this experiment, expecting that it should generate a sorting algorithm.

The structure of the network used in the experiment is illustrated in Figure 5. Most structures are the same as those in Figure 3, but the input data of the recaller differ from key to query. This query is an integer between 0 and $N-1$. Zero is the prompt to let the recaller to output the smallest number among the N input numbers; one is the prompt to let the recaller to output the second smallest number among the N number of inputs; and $N-1$ is the prompt to let the recaller to output the largest number among the N number of inputs. The results of sorting the input values were obtained by arranging the output results. For example, we used the input value of the memorizer for $N=5$, as shown in Figure 6 upper panel. The floating-point numbers are the keys to be sorted, and bold characters are the values. The keys are randomly generated from a uniform distribution with a minimum value of 0 and a maximum value of N. The recaller–output sorted sequence for the input data is depicted in Figure 6 lower panel. The queries to the recaller from top to bottom are 0, 1, 2, 3, and 4.

Learning was conducted in the same manner as described earlier. It was terminated when the validation accuracy reached 0.95. By incrementally increasing the value of N from 2, we checked whether learning could be completed normally. We observed that this could be achieved within 500,000 epochs up to $N=8$, which was consistent with the results presented in the previous section. Although this value of N can vary marginally depending on the stopping condition of learning, discussing whether the value of N is 8, 7, or 9 is meaningless. However, it can be thought that this extent of memory can only be retained by the memorizer–recaller network under the current condition of parameter size and architecture design. As part of the memorizer, we used Formula 2.4. However, we modified the formula as follows:

$$m_t \quad \leftarrow \quad \sigma\left(w_3\left(\frac{p+q}{N}\right)+b_3\right). \tag{2.13}$$

The learning process for generating the sorting algorithm proceeded better in the preliminary experiments. With this type of ingenuity, we can subtly increase the size of N. However, the proposed system does not generate memory vectors suitable for a pre-known size N. Therefore, N should not be included in the arguments of the memorizer. To improve the performance, it is necessary to consider other mechanisms.

Next, for $N=8$, the constructed memorizer–recaller network solved the sorting problems. An output was considered correct only when the order of the elements in the output data from the recaller perfectly matched that in the input data. The test was performed 1,024 times, after which the network could accurately sort in 893 trials. Therefore, the accuracy of the sorting algorithm was 0.872. The generated sorting algorithm cannot provide a perfect answer and can only sort up to 8 numbers. However, it is possible to achieve better results by modulating the network architecture of the

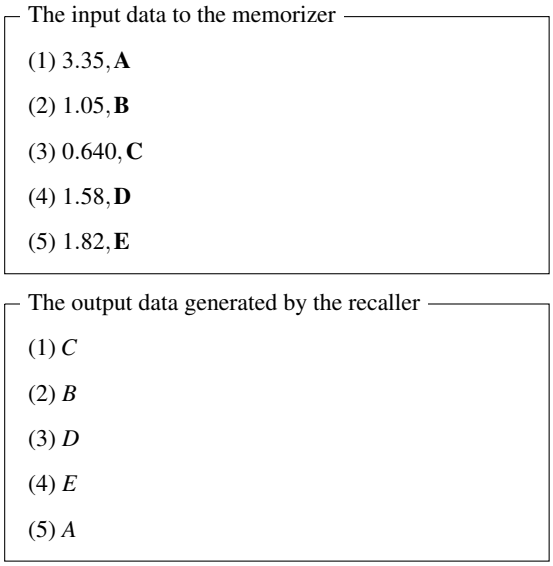

Fig. 6: Example of input and output by a sorting algorithm generated by the memorizer–recaller network. The upper panel includes the input data to the sorting algorithm. The lower panel includes the sorted data by the sorting algorithm.

memorizer–recaller, optimizing the parameter size, or making the learning termination conditions more stringent.

As expected, a sorting algorithm was generated using the memorizer–recaller network. There are various types of sorting algorithms, but the worst time complexity of the comparative sorting method, which is the most standard sorting algorithm, is $O(N \log N)$ when the number of inputs is N. In contrast, the computational complexity of the sorting algorithm generated using the memorizer–recaller is always $O(N)$. This experimental result demonstrates that the memorizer–recaller network can store input information in the memory vector and demonstrated its application to an actual problem.

2.4 Toward Advanced Memory System in Artificial Intelligence

First, we summarize and interpret the terms used thus far. We believe that the long-term memory possessed by several LLMs is a form of external memory. Additionally, we consider both the appendable memory developed in this research and the eventual goal, permanent memory, to be external memory. Long-term memory stores information in the form of natural language, which can be processed by LLMs capable of handling such language. However, it is accompanied by constraints pertaining to the length and quantity of information stored owing to its format. In contrast, the external memory generated by the memorizer is what we call appendable memory, which is a memory vector capable of appending information to it in fixed-sized vectors. Moreover, the recaller can refer to the compressed memory information and restore arbitrary information from it. While the recaller uses these memory vectors to produce information, the LLMs receive lengthy inputs that contain past memories and output natural language. However, we believe that these mechanisms do not include references in their operations.

The aim of this study was to construct a system in which AI can acquire knowledge even after the model has been deployed. Furthermore, we attempted to construct a permanent memory system in which AI can retain an arbitrary amount of information. The proposed memorizer–recaller system can store separately input information in the memory vector of the memorizer and retrieve it using the recaller. The memory vectors generated by the memorizer are external memories that can be saved and utilized subsequently. Additionally, this external memory is a dynamic memory to which new information can be appended. This is where our approach differs from the AI developed thus far. Furthermore, the recaller can restore information from these memory vectors. We attempted to construct a sorting algorithm using this network, which also succeeded in demonstrating that the memorizer could correctly encode the input information into the memory vector and that the recaller could appropriately utilize that information.

The proposed learning method probabilized the training dataset for each epoch and introduced randomness into the learning process. This prevents a model from learning the inherent patterns of the training dataset. While reinforcement learning can be used to make models acquire procedures instead of learning patterns in the data, this study did not use such a mechanism. This aspect is considered a novel idea that can be achieved using random numbers in the learning method. Random numbers have also been used in diffusion models [9] and generative adversarial networks. While those studies did inspire the current work, diffusion models and GANs are indeed learning patterns in the learning data, which makes this approach different.

Although the memorizer–recaller network was originally developed to allow AI to store an arbitrary amount of infor-

mation, the actual amount that can be memorized by the network was only up to 8. Based on the results of preliminary experiments, this number can be improved by increasing the network parameter size. However, this is not the fundamental solution, and further investigations are necessary. The fact that the memorizer–recaller networks do not possess sufficient memory may be related to catastrophic forgetting [10]. Originally, this phenomenon occurred when the models learned one task from another. Because the input information to the proposed memorizer is not related to another task, it does not seem to be related to this phenomenon at first glance. However, the experimental results show that the memorizer–recaller network can completely forget information before a certain point, and thus a type of forgetting is certainly occurring. Several methods were proposed to prevent catastrophic forgetting, including elastic weight consolidation [11] and deep generative reply [12]. We were unable to find a method that could be directly used for the memorizer–recaller network; however, with some ingenuity, it may be possible by applying the methods developed thus far. We plan to examine this issue in the future.

3 Conclusion

This study constructed a system for AI to acquire knowledge even after the model has been deployed, resulting in the creation of the memorizer–recaller network, which can generate appendable memory. Furthermore, the importance of external memory and its application in a dialogue agent to continue learning even after deployment were investigated. We believe that such external memory is already being used by some dialogue agents as long-term memory, albeit partially. However, the current limited external memory is not ideal for continuous learning; therefore, this study proposed appendable memory. The advantage of appendable memory is that new knowledge can be compressed into a finite memory vector, thereby eliminating the need to retain it. The memorizer–recaller system developed in this study successfully stored separate input information in the memory vector of the memorizer and retrieved it using the recaller. In addition, the proposed learning method probabilized the training dataset for each epoch, thereby prohibiting a model from learning patterns inherent in the training dataset. This differentiates our work from previously developed AI systems.

However, despite the original goal of the memorizer–recaller network to store an arbitrary amount of information, the capacity of the memory vector is limited to storing only up to 8 pieces of information. Therefore, further investigations are required to address this issue. Despite the preliminary results of this study, the nature of appendable memory is different from that of existing memory, and the method to train memorizer–recaller is fundamentally different from previously implemented neural network learning methods; thus, this approach is considered groundbreaking. In the future, we will continue our study to ensure that the proposed method is considered foundational for building an AI capable of storing information in finite-sized data and utilizing it freely.

Funding

This work was supported in part by the Top Global University Project from the Ministry of Education, Culture, Sports, Science, and Technology of Japan (MEXT).

REFERENCES

[1] Kazunori D Yamada, Samy Baladram, and Fangzhou Lin. Progress in research on implementing machine consciousness. *Interdisciplinary Information Sciences*, 2022.
[2] Braden Hancock, Antoine Bordes, Pierre-Emmanuel Mazare, and Jason Weston. Learning from dialogue after deployment: Feed yourself, chatbot! *Annual Meeting of the Association for Computational Linguistics*, 2019.
[3] OpenAI. Gpt-4 technical report. *arXiv*, 2023.
[4] Jing Xu, Arthur Szlam, and Jason Weston. Beyond goldfish memory: Long-term open-domain conversation. *arXiv*, 2021.
[5] John R. Searle. Minds, brains, and programs. *Behavioral and Brain Sciences*, 1980.
[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Conference on Neural Information Processing Systems*, 2017.
[7] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. *International Conference on Machine Learning*, 2013.
[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2015.
[9] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Conference on Neural Information Processing Systems*, 2020.
[10] Vinay Venkatesh Ramasesh, Ethan Dyer, and Maithra Raghu. Anatomy of catastrophic forgetting: Hidden representations and task semantics. *International Conference on Learning Representations*, 2021.
[11] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 2017.
[12] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *Conference on Neural Information Processing Systems*, 2017.