applied sciences

MDPI

Article

A Conversation History-Based Q&A Cache Mechanism for Multi-Layered Chatbot Services

Ozoda Makhkamova and Doohyun Kim *

Intelligent Things Lab, Department of Computer Science, Konkuk University, 120 Neungdong-ro, Gwangjin-gu, Seoul 05029, Korea; crchoc@gmail.com
* Correspondence: doohyun@konkuk.ac.kr

Abstract: Chatbot technologies have made our lives easier. To create a chatbot with high intelligence, a significant amount of knowledge processing is required. However, this can slow down the reaction time; hence, a mechanism to enable a quick response is needed. This paper proposes a cache mechanism to improve the response time of the chatbot service; while the cache in CPU utilizes the locality of references within binary code executions, our cache mechanism for chatbots uses the frequency and relevance information which potentially exists within the set of Q&A pairs. The proposed idea is to enable the broker in a multi-layered structure to analyze and store the keyword-wise relevance of the set of Q&A pairs from chatbots. In addition, the cache mechanism accumulates the frequency of the input questions by monitoring the conversation history. When a cache miss occurs, the broker selects a chatbot according to the frequency and relevance, and then delivers the query to the selected chatbot to obtain a response for answer. This mechanism showed a significant increase in the cache hit ratio as well as an improvement in the average response time.

Keywords: chatbot; multi-layer service; cache mechanism; response time; chatbot cache

check for updates

Citation: Makhkamova, O.; Kim, D. A Conversation History-Based Q&A Cache Mechanism for Multi-Layered Chatbot Services. *Appl. Sci.* **2021**, *11*, 9981. https://doi.org/10.3390/app11219981

Academic Editor: Alessandro Di Nuovo

Received: 20 August 2021
Accepted: 21 October 2021
Published: 25 October 2021

1. Introduction

Due to development in Internet technologies, online learning has grown from Internet-propelled distance education to online higher education enrollments. Online education has such benefits as saving money and time, choosing an area of interest and a course of study individually [1]. One of the ways of learning online is using video-based online courses (MOOC, Coursera, edX, and others) [2–4]. The demand for online lectures is increasing, and accordingly, the combination of MOOC and chatbot is becoming more useful [5]. It helps overcome one of the limitations and weaknesses of MOOC, which is taking a long time to receive feedback [6]. One of the examples of the combination of MOOC and chatbot systems is a video tutoring assistant for online video tutoring [7]. The video tutoring assistant for online video tutoring is a chatbot service that plays the role of an assistant and provides real-time feedback to user questions. Thus, the main goal of this chatbot is to answer specific user questions to shorten the time it takes to receive feedback from a lecturer.

In this paper, a multi-layered multi-chatbot system architecture is proposed for the video tutoring assistant to improve the coverage of Q&A that the system deals with. It uses multiple chatbots for each video instead of using a single chatbot. This strategy, however, may decrease the speed of the system, and a cache mechanism is necessarily required consequently to overcome such decline of the speed. The concept of layers comes from edge computing in cloud technologies. This technology helps to ease the system and its process. The cache mechanism is one of the ways to use edge computing.

The cache in CPU utilizes the locality of references within binary code executions [8]; web cache policy is created based on the location of the documents [9]. The cache mechanism proposed in this paper uses the frequency and relevance information which potentially exists within the set of Q&A pairs.

The remainder of this paper is organized as follows. Section 2 introduces the problem definition and previous approaches to solve this problem. In Section 3, we introduce the architecture and algorithm of the caching mechanism proposed in this paper. In Section 4, we discuss experiments and the experimental results. Section 5 concludes the paper with a summary of our main findings and contributions.

2. Related Work

A recent study on chatbot technologies shows the usage of different algorithms and models. M.Boyanov, I. Koychev introduced the chatbot that uses the seq2seq algorithm to give an answer to the users of Web forums [10]. The authors used web forum data and preprocessed them before training the model. It showed good results on answering correctly to questions asked in the forum and conversationally styled questions. The main solution of this paper is the selection of sentences from a Q&A pair-by-word product over averaged word embedding representation. In the case of [11], the chatbot system was created with DialogFlow and BERT algorithm. This chatbot was created for the Georgia Southern University homepage so the students can ask questions related to the university and its staff.

Nowadays, the usage of chatbot technologies and online learning is growing. As was mentioned before, the main goal of the video tutoring assistant is to shorten the time of getting feedback from the lecturer and increasing student's satisfaction and attendance until the end of the course. A. Espasa and J. Meneses presented work on feedback in an online learning environment [12]. The authors show an association between feedback and student performance. This brings us to the relevance of feedback in self-regulatory competencies within online learning and learning progress. This is the reason the authors of [7] considered combining online video learning with chatbot technologies. The other combination of the online video learning platform and chatbots is the usage of chatbot XiaoMu in XuetangX [13,14]. J. Tang presents a chatbot system for XuetangX with three main functions: course recommendation, question answering, and automated video navigation. This chatbot uses a knowledge graph to create a recommendation system and graph for questions.

We suggest using the multi-layered structure in the video tutoring assistant [7] because it uses multiple chatbots for one video and must connect with chatbots every time the user asks the question. This proposal came up from the idea of edge computing in cloud technologies. Edge computing plays a big role in reducing networking pressure and service response time, improving the performance of low-power devices [15]. The authors of [16] introduce the response time as one of the attributes that play a role in users' satisfaction. Bangla chatbot uses a knowledge-based tree structure to reduce steps of finding an answer in the whole conversation [17]. Our approach reduces steps of getting an answer from the chatbot with the concept of edge computing.

3. Problem Definition and Our Approach

Generally, the cache mechanism of the CPU is optimized for binary code execution based on the fact that the implementation of binary code has a high reference to the locality of the code [18]. Web cache uses the locality of the documents and replaces the least recently used data [19]. In other words, the code has an address, and that address makes the code implementation local; thus, it is assumed that caching nearby code will increase the hit ratio [20,21]. The hit ratio shows how successfully the cache retrieves data and content, and the miss ratio is the number of times the system requested data and content that is not cached. The purpose of the cache is to increase the hit ratio and decrease the miss ratio (1).

$$Hit = \frac{cacheratio}{cacheratio + missratio}, \tag{1}$$

To achieve the highest performance, cache misses should occur as seldom as possible. Considering that the cache memory is much smaller than the RAM, it is not easy to achieve a high cache hit ratio. This is the reason the main task of the cache is to load the cache memory with the necessary data and promptly remove data that are no longer required. The simplest algorithm for loading data into the cache is based on the assumption that data from the main memory are processed sequentially, in ascending order of the addresses.

The video tutoring assistant consists of three main parts in different layers. The first layer of the system is the broker. This layer handles the cache mechanism and chatbot management. The next layer is the chatbot layer, which connects multiple chatbots that are registered in the system. Various chatbots may be prepared by different developers with differentiated question and answer (Q&A) sets for the video. The broker connects with chatbots and plays the role of a bridge between the user and chatbots.

This broker can manage the conversation history of the users and provides a personalized cache memory. In the case of LMS (learning management system), one subject has a fixed number of students, and the information of these students is advantageous for the personalization of brokers. The number of brokers equaled to the number of students of the subject makes the management of conversation history easier and convenient. The usage of the broker for each student helps in the personalization of conversation history and the cache memory. These personalized brokers can share frequency information within the whole conversation history of the class. However, it is one of the ideas to show the extensibility of broker-based architecture.

The last layer is a deep learning-based chatbot. Besides, the user terminal is a webpage for the video tutoring assistant that allows the user to have a conversation with the system. It allows users to ask the broker possible questions that may arise while watching the video. Figure 1 shows our multi-layered, multi-chatbot system architecture for a video tutoring assistant.

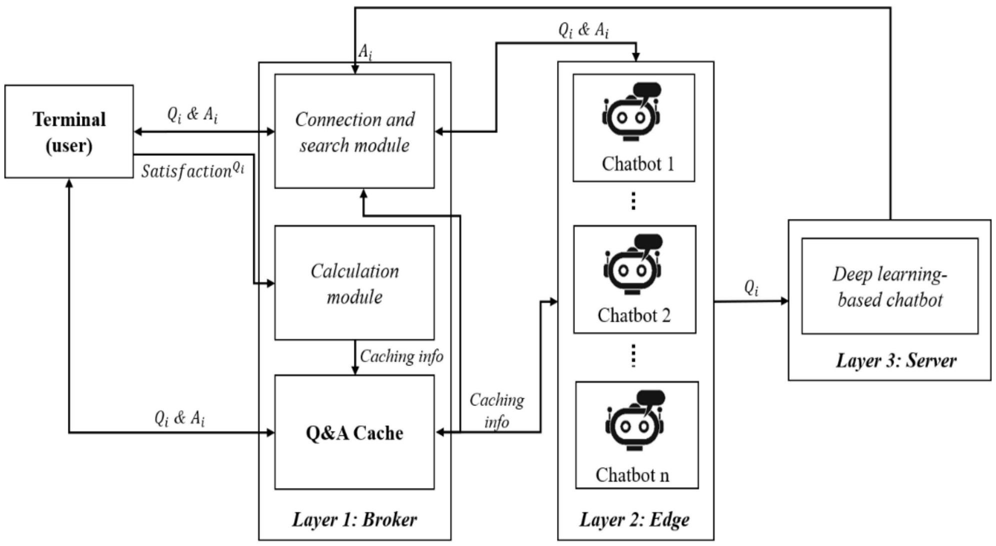


Figure 1. System architecture of video tutoring assistant chatbot.

In this paper, we propose a caching mechanism for the broker in a multi-chatbot video tutoring system using the frequency and relevance based on the conversation history and keywords. The purpose of the caching mechanism proposed for this system is to enhance its responsiveness by caching relevant, most likely to be asked Q&A sets. The cache mechanism analyzes the relevance between Q&A pairs. Next, the frequency information of these questions is accumulated continuously as it arrives. In other words, the Q&A sets relevant, with high frequency to the question asked at this moment, are cached.

The mechanism in (2) finds the set of Q&A pairs with the maximum hit ratio of the cache. In this study, frequency and relevance between Q&A sets helps to achieve the maximum hit ratio.

$$\hat{k} = argmax_k(HitRatio(S_k)) \tag{2}$$

where S_k denotes a set of Q&A pairs that could be cached and $\hat{k}$ is the id of the set with the maximum hit ratio.

The cache mechanism in (3) chooses the most relevant Q&A pairs from the unit to maximize the hit ratio:

$$RelQs = [RQ_0^i,\ RQ_1^i,\ \ldots,\ RQ_j^i]\ for\ Q_i \quad (3)$$

where *i* is the index of the current Q, *j* is the index of relevant Q&A sets, *RQ* is the relevant Q&A pair, and *RelQs* is the list of relevant Q&A pairs. Then, the cache mechanism in (4) selects the most frequently asked Q&A pairs based on the unit

$$FreqQs = [FrQ_0^i,\ FrQ_1^i,\ \ldots,\ FrQ_j^i]\ for\ Q_i \quad (4)$$

where *i* is the index of the currently asked question, *j* is the number of relevant questions, *FrQ* is the frequency of the chosen relevant question, and *FreqQs* is the list of frequencies of the relevant Q&A pairs.

4. Cache Mechanism

The cache mechanism aims to bring Q&A sets often asked of chatbots to the cache layer. This approach is intended to improve the response time of the broker and ease the system processes.

4.1. Cache Mechanism Architecture

The cache mechanism has three main parts. The first part is the reply module, which is responsible for replying to the query of the user if it is already cached. The second part is the keyword extraction module [22], which extracts keywords for all Q&A sets of the chatbots and sends this information to the third module. The third module, which manages frequency and relevance, creates a relevance graph of the Q&A sets based on the extracted keywords. In addition, this module is responsible for creating a frequency graph of the Q&A sets (Figure 2).

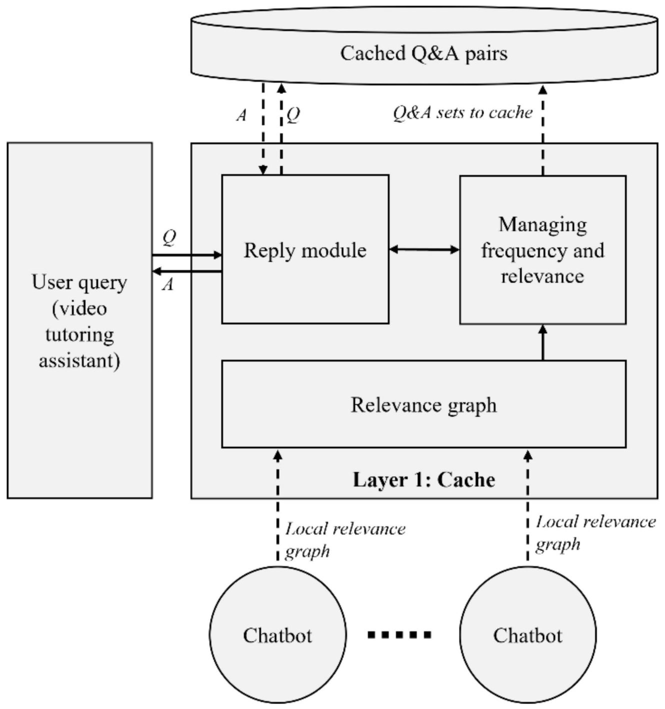


Figure 2. Cache mechanism architecture.

4.1.1. Reply Module

The reply module of the mechanism receives user queries and returns matched answers to the user. It uses cache memory to decrease response time. The module uses the AIML engine [23] to match the question from the user with the cached Q&A sets. The AIML engine is an old one, but it is light, which is a good characteristic for using as a cache-matching engine. In this system, the cache is a micro-chatbot, and it must have the minimum functionality of the chatbot. That is why it is enough to use AIML in the cache. AIML provides tags and has specific rules to match questions. However, increasing matching flexibility by generalizing questions using wildcards in AIML should be a separate study. If the query is cached, the module replies to the user. If the query is not in the cache, the module sends the question to a predicted chatbot to obtain an answer to return to the user.

4.1.2. Keyword Extraction Module

The keyword extraction module uses the TextRank [24] algorithm to extract the top keywords for all Q&A pairs to create a relevance graph of the Q&A sets. This graph indicates the relevance among Q&A pairs by ranking them based on the number of keywords that they share. The nodes of the graph are Q&A pairs, and the edges are the relevance scores of the Q&A based on the number of shared keywords. Table 1 shows the top keyword examples from the keyword extraction by the TextRank algorithm from Q&A sets about computers and AI.

Table 1. Example of top keywords.

Keyword	Keyword	Keyword	Keyword
computer	program	system	object
Java	software	data	network
language	Windows	internet	HTML
AIML	category	methods	source
robot	pattern	Linux	tags

One of the limitations of using the relevance graph of the Q&A sets is a requirement of whole Q&A sets of all chatbots, and this requirement makes chatbot developers send Q&A datasets to the system. The request of whole Q&A sets of chatbots might force chatbot developers to open their knowledge, and they would not want to give it for free. One of the possible ways to overcome this limitation is to ask for a local relevance graph from each chatbot and merge such pieces into a whole relevance graph. This local relevance graph does not contain Q&A texts but consists of only Q&A id and the chatbot id with the name of the chatbot.

4.1.3. Frequency and Relevance Management Module

As mentioned above, the keywords extracted by the keyword extraction module are used to create the Q&A graph based on the mutual relevancies of all Q&A pairs. The usage of keywords helps to calculate the relevance of the Q&A pairs. In this study, Q&A pairs with the same keywords are considered relevant to each other. Figure 3 shows an example of the Q&A relevance.

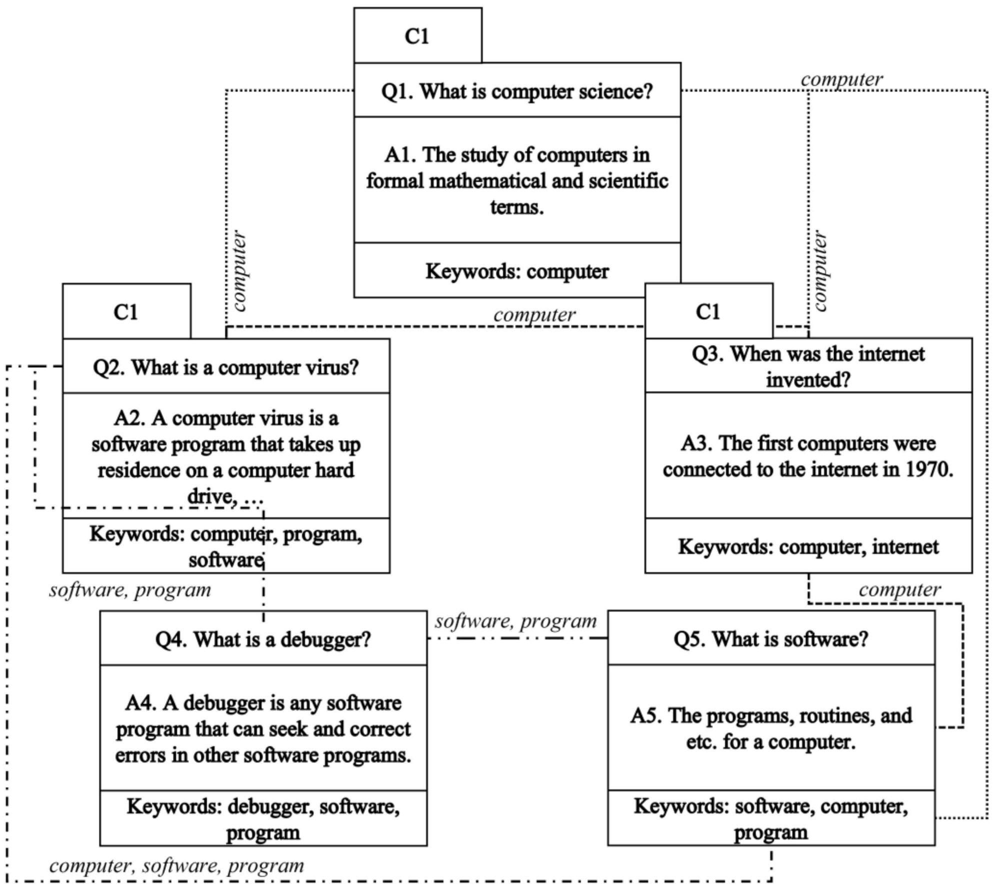

C1
Q1. What is computer science?
A1. The study of computers in formal mathematical and scientific terms.
Keywords: computer
C1
Q2. What is a computer virus?
A2. A computer virus is a software program that takes up residence on a computer hard drive, …
Keywords: computer, program, software
C1
Q3. When was the internet invented?
A3. The first computers were connected to the internet in 1970.
Keywords: computer, internet
Q4. What is a debugger?
A4. A debugger is any software program that can seek and correct errors in other software programs.
Keywords: debugger, software, program
Q5. What is software?
A5. The programs, routines, and etc. for a computer.
Keywords: software, computer, program
computer
computer
computer
computer
software, program
software, program
computer
computer, software, program

Figure 3. Example of relevancies among Q&A pairs.

The meaning of this graph is as follows:

The keyword of $Q_1A_1C_1$ is "computer," and this Q&A pair has relevance to questions from the related questions list *RelQs = [*$Q_2A_2C_1$, $Q_3A_3C_1$, $Q_5A_5C_3$*]*.

The $Q_2A_2C_1$ pair has keywords "computer," "program," and "software," and the relevant Q&A pairs are *RelQs = [*$Q_1A_1C_1$, $Q_3A_3C_1$, $Q_4A_4C_2$, $Q_5A_5C_3$*]*.

In the case of $Q_3A_3C_1$, the keywords are "computer" and "internet," and *RelQs = [*$Q_1A_1C_1$, $Q_2A_2C_1$, $Q_5A_5C_3$*]*.

The $Q_4A_4C_2$ pair's keywords are "debugger," "software," and "program," and *RelQs = [*$Q_2A_2C_1$, $Q_5A_5C_3$*]*.

The $Q_5A_5C_3$ pair's keywords are "software," "computer," and "program," and *RelQs = [*$Q_1A_1C_1$, $Q_2A_2C_1$, $Q_3A_3C_1$, $Q_4A_4C_2$*]*.

The concept of relevance proposed in this paper depends on the number of similar keywords between Q&A pairs. In the case of [25], the policy of a cache mechanism depends on the document semantics. The algorithm of this paper caches semantically closed documents to the current page and removes the least frequently used documents. The proposed cache in this paper uses similar keywords in Q&A pairs to create relevance. This method might have limitations, but it is one of the easiest ways to make the cache work. Figure 4 shows the representation of the relevance graph. This graph has the id of Q&A (Q_1A_1, Q_2A_2, Q_3A_3, Q_4A_4, Q_5A_5) and the chatbot id (C_1, C_2, C_3) that sent this information to the cache.

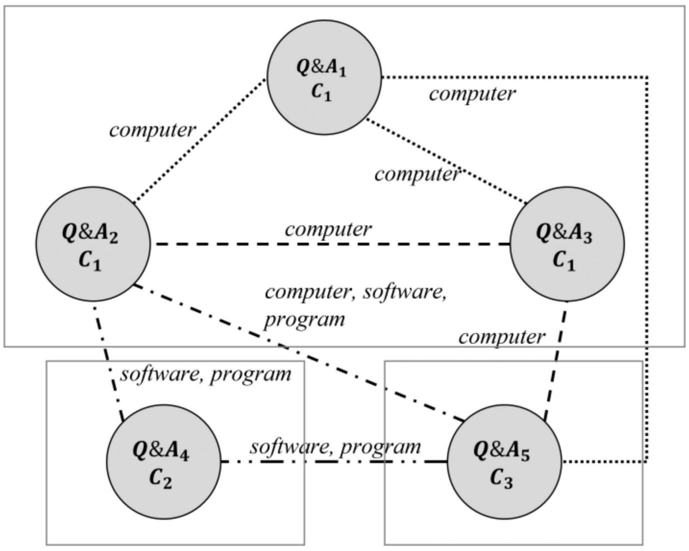


Figure 4. Example of Q&A relevance graph.

The frequency in this mechanism depends on the conversation history of the chatbot. Here the frequency calculation works based on the previous and present questions. In other words, the frequency of a question is the number of times each question in the Q&A dataset is followed immediately by this question. This concept is different from the context view of NLP; in fact, handling the context of the NLP view is beyond the scope of this paper. Nevertheless, the method used in this paper, that is, the method based on the occurrences of previous questions in the conversation history, is considered to be engineeringly suitable for the cache mechanism.

The frequency of Q&A pairs is computed using conversation history. Table 2 presents an example of a Q&A set frequency graph showing how many times each question was called after other questions from the Q&A dataset.

Table 2. Exampl e of Q&A set occurrence dependency table.

	Q_1	Q_2	Q_3	Q_4	Q_5
$\mathbf{Q_1}$	0	1	5	0	2
$\mathbf{Q_2}$	6	0	5	1	0
$\mathbf{Q_3}$	5	4	0	2	0
$\mathbf{Q_4}$	1	1	3	0	3
$\mathbf{Q_5}$	3	8	0	2	0

For example, after Q_1, [Q_3] was asked five times, [Q_4] was asked three times, [Q_2] only once, and Q_1 never (as it is itself). Another example is Q_5, where [Q_2] was asked eight times after Q_5, [Q_1] three times, [Q_4] two times, and [Q_3] was never asked. This table is updated every time a new question is asked. This process follows the next equation:

$$F(i,\ j) += 1,\ if\ Q_j == Prev(Q_i) \tag{5}$$

where Q_i is the current question, Q_j is the previous question, $Prev(Q_i)$ is the previous question, and $F(i,\ j)$ is the number of times the current question (Q_i) was asked immediately after the previous question (Q_j).

The table can be represented by a graph, as shown in Figure 5. The number next to a dashed arrow ("- ->") shows the number of times a specific question was asked after Q_5 (Q_1: three times, Q_2: eight times, Q_4: two times), and the number next to a solid arrow ("→") shows how many times a specific question was called after Q_1 (Q_2: one time, Q_3: five times, Q_5: two times).

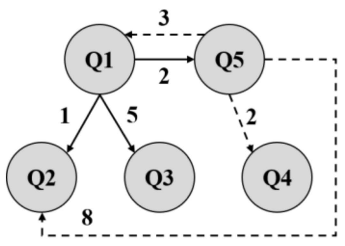


Figure 5. Example of Q&A set frequency graph (Q_1, Q_5).

4.2. Cache Mechanism Algorithm

The proposed algorithm saves some of the Q&A sets from the chatbot data to generate responses directly from the broker without connecting to the chatbots. The usage of frequency and relevance enables the algorithm to cache effectively. Algorithm 1 shows the cache mechanism algorithm.

Algorithm 1 Cache Mechanism.

```
1   input: $Q_i$                                      #user's question
2   $RelQs_i$ = RelQ [$Q_i$Ck]                         #related questions from
3   #correspondent chatbot
4   for RQ in $RelQs_i$:
5   $FreqQs_i$ [RQ] = FreqQ [$Q_iC_k$] [RQ]            #frequency of each RQ from
6   #correspondent chatbot
7   end for
7   for RQ in $RelQs_i$:
9   if RQ not in cache:
10  cache.append(RQ)
11  end if
12  end for
13  if len(cache) > size:                             #size of cache
14  for CQ in cache:                                  #cached question
15  $FreqCQs_i$ [CQ] =  FreqQ [$Q_iC_k$] [CQ]:         #frequency of each CQ
16  sorted($FreqCQs_i$[CQ].keys(), FreqCQs $_i$[CQ].values())
17  end for
    end if
    cache = $FreqCQs_i$.keys()[:size]
```

The input of this algorithm is a question from the chatbot service (Q_i). The algorithm uses the keyword-based relevance graph (*RelQ*) to obtain the next Q&A pairs and save them in $RelQs_i$ (C_k is the id of the chatbot to get the required related questions). As Q&A sets are cached based on relevance and frequency, it is necessary to obtain the frequency of the relevant Q&A pairs ($FreqQ_i$) based on the conversation history. In other words, the frequency graph (*FreqQ*) saves information on how many times each relevant question was asked after Q_i. The frequency helps the algorithm to sort relevant questions in reverse order to obtain the top Q&A sets that must be cached. The same is implemented with previously cached Q&A sets (*CQ*). The algorithm obtains the frequency of the previously cached Q&A sets ($FreqCQ_i$) after Q_i from *FreqQ* and unites it with $FreqQ_i$. Then, only the top number (*size*) of Q&A sets are cached.

4.3. Handling Cache-Miss

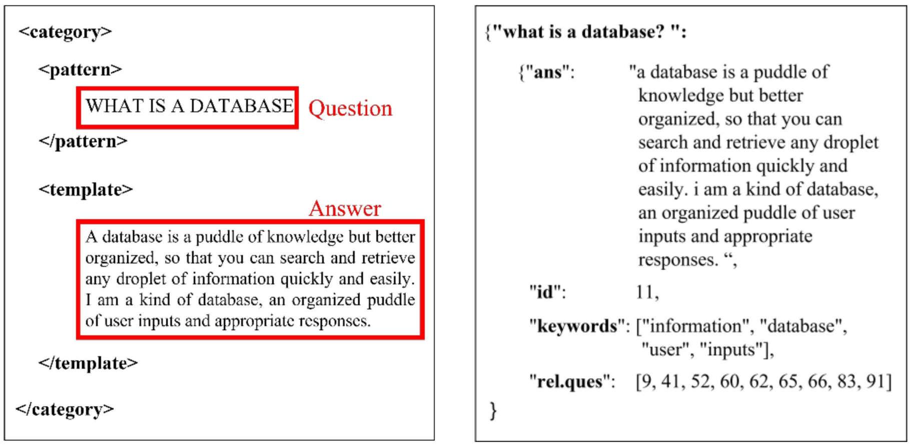
The system does not have the questions in the cache always, but it has cache misses too. In this case, the system requires the answer from the chatbots. The correspondent chatbot is selected based on the relevance graph. The system finds the node of the Q&A pair that has the highest number of keywords registered in the edge. Then it connects to the chatbot mentioned on the node and requires the answer from the chatbot.

5. Experiments

5.1. Data Preparation

The data required for the experiments needed some modification to suit the form of the experiments. These experiments estimated the caching mechanism by maximizing the hit ratio. The data in this experiment were obtained from the AIML archive for "Computers" and "Artificial intelligence (AI)" [25].

The data consisted of 200 Q&A pairs in the AIML file. The data needed to be modified to be used to obtain relevant Q&A pairs and frequencies. Figure 6a shows an example of a Q&A pair in an AIML file. The question is indicated by the tags <pattern> and </pattern>, and the answer is indicated by <template> and </template>. Figure 6b is an example of a modified Q&A pair and it is a json format. This example was created based on data from the AIML file, with keywords extracted by the TextRank algorithm. The cache mechanism uses data in this structure, which saves information about the answer in "ans," the id of the Q&A pair in "id," keywords of the Q&A pair extracted with the TextRank algorithm in "keywords," and relevant Q&A pairs in "rel.ques." The "*keyword*" part helps to find the "*rel.ques*" information, which is used to find the Q&A sets to cache.



```
<category>
  <pattern>
      WHAT IS A DATABASE
  </pattern>
  <template>
      A database is a puddle of knowledge but better
      organized, so that you can search and retrieve
      any droplet of information quickly and easily.
      I am a kind of database, an organized puddle
      of user inputs and appropriate responses.
  </template>
</category>
```

Question
Answer

```
{"what is a database? ":
   {"ans":        "a database is a puddle of
                   knowledge but better
                   organized, so that you can
                   search and retrieve any droplet
                   of information quickly and
                   easily. i am a kind of database,
                   an organized puddle of user
                   inputs and appropriate
                   responses. ",
     "id":         11,
     "keywords":  ["information", "database",
                   "user", "inputs"],
     "rel.ques":  [9, 41, 52, 60, 62, 65, 66, 83, 91]
}
```

(**a**) (**b**)

Figure 6. Example of Q&A pair in AIML file (**a**) and modified Q&A pair (**b**).

5.2. Experimental Setup

The cache mechanism proposed in this paper was implemented in Python. All experiments were conducted using a processor with a desktop and the following specifications:

CPU: Intel® Core™ i5-8500 CPU @ 3.00GHz
RAM: 8.00 GB
OS: Windows
Python: 3.6.7

5.3. Comparison Experiment and Results

The experiment included three types of sequences, namely random, mixed, and related, and the following five cache sizes: 10, 15, 20, 25, 30. Asking random questions with no relevance or frequency is a random sequence. A mixed sequence consists of 4000 random questions followed by 6000 questions relevant to each other. A related sequence is a set of Q&A pairs stored in order of relevance to each other to have humanlike and content-based conversations. The cache size is the number of Q&A pairs that can be saved in the cache. The number of Q&A pairs is different for each case. There were four experimental cases with the same cache size parameter and sequences, but the size of the Q&A pairs was different. Case 1 had 50 Q&A pairs, Case 2 had 100, Case 3 had 150, and Case 4 had 200. Table 3 shows all the parameters for the experiment.

Table 3. Information on experimental cases.

Case	Q&A Set	Cache Size	Questions Sequence	Steps
Case 1	50	10, 15, 20, 25, 30	Random, mixed, related	10,000
Case 2	100	10, 15, 20, 25, 30	Random, mixed, related	10,000
Case 3	150	10, 15, 20, 25, 30	Random, mixed, related	10,000
Case 4	200	10, 15, 20, 25, 30	Random, mixed, related	10,000

The advantage of the proposed method is analyzing relevance among keywords in Q&A sets and frequency based on the conversational history in place of code locality. As was mentioned before, the CPU cache uses code locality to add instructions and data to the cache. However, Q&A sets of the chatbots has no locality information and the usage of the frequency and relevance to cache questions might be one of the possible ways.

This experiment aimed to show how well the proposed cache mechanism worked and the relationship between cache size and Q&A set size. A lower number of Q&A pairs with a large cache size might show a high hit ratio, but a large Q&A set with a small cache size would produce a lower hit ratio. This experiment was intended to show the consistency between Q&A set number and cache size and the reduction in the response time.

5.3.1. Q&A Set Size and Cache Size Relationship

Case 1 (C1) had 50 Q&A pairs; the cache size was equal to 10, 15, 20, 25, and 30, and the caching sequence was random, mixed, and related. Figure 7a–c shows the results of C1 in random, mixed, and related sequences, respectively. These results show how the cache size and order affect the hit ratio. The random sequence does not show good hit ratios, regardless of the cache size (Figure 7a). The C1 results for the mixed-question sequence show how the hit ratio became higher after related questions were asked after the first 4000 steps (Figure 7b). The related-question sequence in C1 produced the best results with cache size 30 (Figure 7c). However, this result cannot be considered a good result because the cache size is more than 50% of the entire dataset. The bar graph of all cache sizes in Figure 7d shows the amount of difference depending on the question sequence.

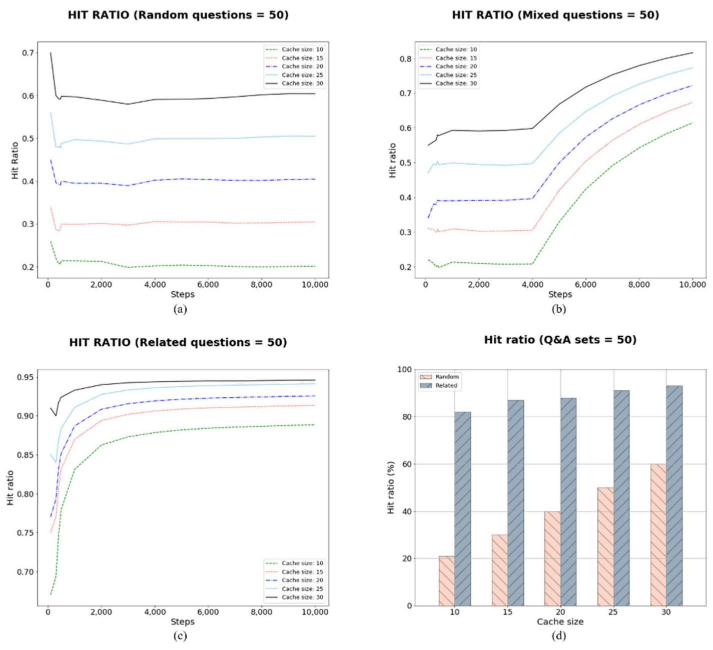


Figure 7. Experimental results of C1. (**a**)—results of C1 with the random sequence; (**b**)—results of C1 with the mixed questions sequence; (**c**)—results of C1 with the related questions sequence; (**d**)—the overall results of C1.

The results for C2 with all cache sizes and random-sequence questions show almost the same results as C1. The mixed-sequence results show improvement after related questions were asked. The sequence of related questions had a high hit ratio, particularly with cache sizes equal to 25 and 30. The random sequence had a hit ratio less than 30%. The hit ratio of the mixed-question sequence was less than 40%. The highest hit ratio had cache size 30, and the lowest hit ratio had cache size 10.

In the case of C3, the experiment produced approximately the same results, with an insignificant difference between the hit ratio averages. As the number of questions grew, the hit ratio decreased. With the mixed-question sequence, the hit ratio increased after step 4000, and with the related-question sequence, the hit ratio also increased. In the related-sequence test of C3, all cache sizes produced a hit ratio greater than 80%.

The random sequence in C4 produced the same low hit ratio as in the other cases (Figure 8a). The mixed-question sequence produced a low hit ratio before the related questions were asked, after which the hit ratio increased. The related-question sequence of C4 produced an increasing hit ratio for all cache sizes, as shown in Figure 8c. The comparison of all cache sizes in C4 shows a significant difference between the related and random sequences (Figure 8d). The first produced an average hit ratio slightly below 80%, but the other one produced a hit ratio less than 30%.

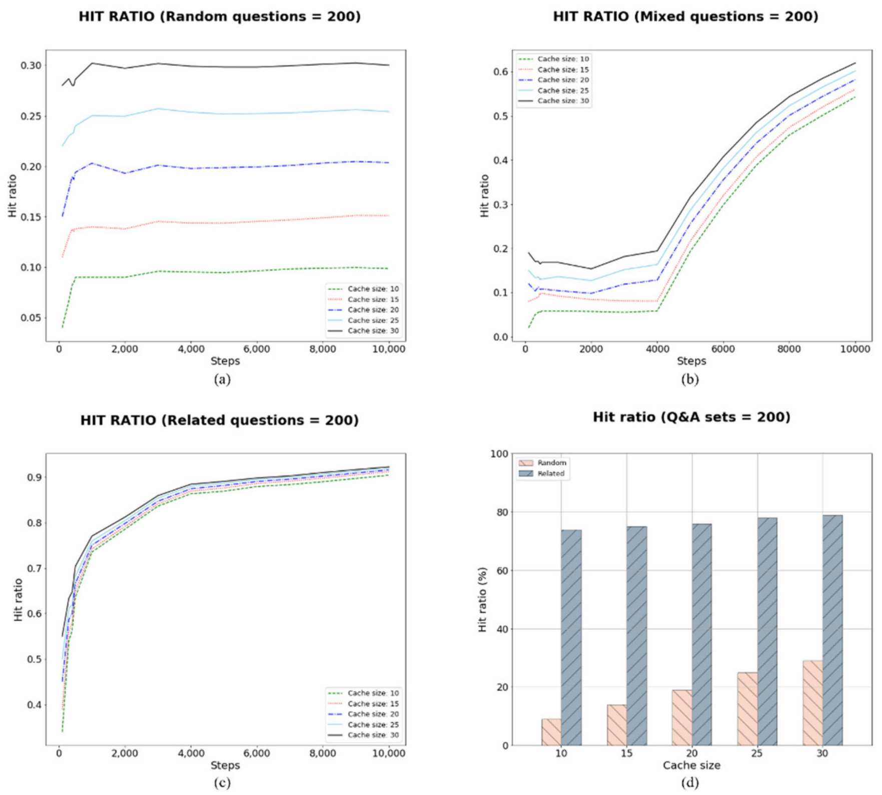


Figure 8. Experimental results of C4. (**a**)—results of C2 with the random sequence; (**b**)—results of C2 with the mixed questions sequence; (**c**)—results of C2 with the related questions sequence; (**d**)—the overall results of C2.

The four experimental cases used the same parameters, with the only difference being the number of questions. The experimental results showed that the sequence of the questions affected the hit ratio. A logical conversation with a user would create a better frequency graph of Q&A pairs than random conversations would. The experiment demonstrated the difference between cache sizes and the number of Q&A sets. Table 4 shows the average hit ratio of all cases and all cache sizes in random and related sequence of questions.

Table 4. Average hit ratio of all cases in random and related-question sequence.

	Sequence	**Cache Size**				
		10	**15**	**20**	**25**	**30**
Case 1 (50)	Random	21	30	40	50	60
	Related	82	87	88	91	93
Case 2 (100)	Random	11	16	22	27	32
	Related	80	82	83	85	86
Case 3 (150)	Random	7	11	7	17	20
	Related	77	78	82	81	82
Case 4 (200)	Random	9	14	19	25	29
	Related	74	75	76	78	79
Total	Random	12	17.75	22	50	35.25
	Related	78.25	80.5	76.75	79	81

The results in the table show how the sequence of questions produces the difference in hit ratio. As a random sequence has no meaning, the hit ratio shows low results. In the case of C2, C3, and C4, the hit ratio in the related sequence increased more than two times compared to the random sequence. It shows the high average in a related sequence, which means the system will show the high hit ratio as the user might have a meaningful and context-based conversation. Table 4 shows the high average hit ratio for cache size 30. The average hit ratio decreases as the number of Q&A pairs increases. In Case 1, cache size 30 included 60% of the Q&A pairs, while cache size 30 for Case 2 included only 30% of the pairs. For Case 3, cache size 30 included 20%, and for Case 4, only 15% of the pairs.

5.3.2. Response Time

Response time in this study is the round-trip time from the user terminal to the system and back to the user. Round Trip Time (RTT) is the length of time it takes for a data packet to be sent to a destination plus the time it takes for an acknowledgment of that packet to be received back at the origin. The aim of this study was to use the cache mechanism to decrease the RTT from “user → broker → chatbot → broker → user” to “user → broker (cache) → user” (Figure 9).

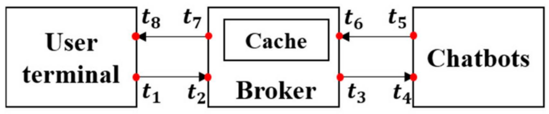


Figure 9. Response time of the system process.

Figure 9 shows the RTT of a system without a cache mechanism. The RTT in this case is calculated by (6):

$$RTT_{chatbot} = (t_2 - t_1) + (t_4 - t_3) + (t_6 - t_5) + (t_8 - t_7) \tag{6}$$

In the case of a system with a cache mechanism, RTT is calculated by the following equation:

$$RTT_{cache} = (t_8 - t_7) + (t_4 - t_3). \tag{7}$$

Figure 10 shows the difference between response times for a question in the cache and the question from the chatbot itself. These results show that the main goal of this study was achieved, which was to shorten the response time of chatbot services by caching Q&A sets. The environment for the calculation of response times consisted of three different terminals. The first was the terminal of the user, who was sending questions to the broker. The broker terminal was responsible for caching Q&A sets and answering the user's question from the cache or through the chatbot. The final terminal was the chatbot terminal, which answered the user's questions that were not cached. Two questions were asked 500 times each. One question was already cached, and another question was not cached yet. In the case of a cached question, the user's question was sent to the broker, and the answer was shown to the user immediately from the cache. However, in the case of a question that was not cached, the system sent the user's question to the broker, and the broker sent it to the chatbot, where it was matched with Q&A pairs in the chatbot database, and the answer was returned to the user from the chatbot through the broker. In Figure 10, the blue line shows response times of the system with answers from the chatbot ($RTT_{chatbot}$). The red line shows response times of the system with answers from the cache (RTT_{cache}). The difference in response time is 2–3 times, as the average response times for questions that are not cached is $RTT_{chatbot} = 2.9\ ms$, whereas the average response time for cached questions is $RTT_{cache} = 0.9\ ms$.

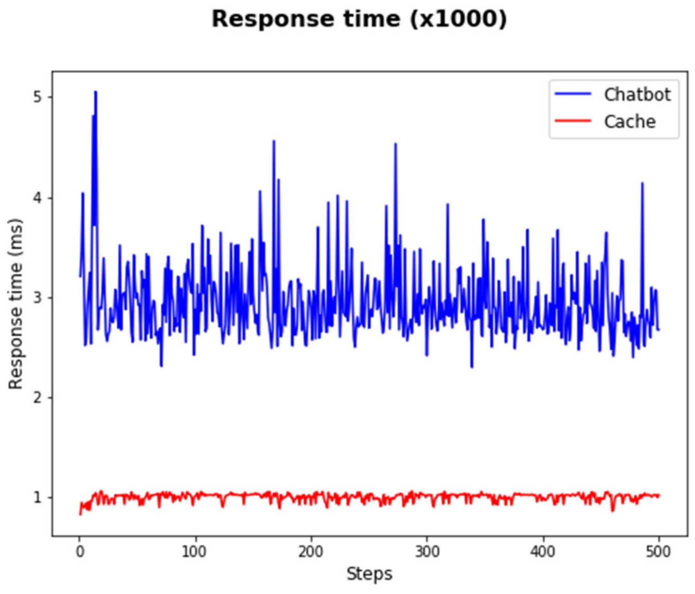


Figure 10. Experimental results for response time.

The experimental results showed what we expected to see. The experiment showed that the Q&A set number and cache size are consistent. As the Q&A set number increases, the hit ratio decreases. An increase in cache size produced an increase in the average hit ratio. As people ask questions based on the context of the conversation, the results using related sequences show the proposed cache mechanism is an effective one to use in the system. The presented cache mechanism can lighten system processes and reduce the response time of the chatbot system. To save all relevant Q&A pairs in the cache is ideal. However, cache size is limited, which is the reason to cache Q&A sets that have higher frequencies. In the results, as Q&A pairs with high frequencies were more likely to be asked, the hit ratio did not significantly decrease.

6. Conclusions

The proposed cache mechanism is responsible for improving the response time of a multi-layered, multi-chatbot system. It is included in the multi-layered broker system to help the broker provide faster answers to user questions. The caching mechanism aims to store often-asked relevant Q&A sets based on conversational history and keywords. Using the cache mechanism lightens system processes and improves response time by sending immediate answers to questions from the cache memory in the broker.

Experiments with four cases having different numbers of Q&A pairs were conducted to show the performance of the presented cache mechanism. The results of this experiment show that the cache mechanism performed well. This mechanism works based on the relevance of Q&A pairs based on the same keywords and frequency based on the conversational history. The experiment achieved up to a 94.00% hit ratio in contextual-question sequences with 100 Q&A sets and cache size 30, which was better than 66.70% in random-question sequences with 50 Q&A sets and cache size 30.

As was mentioned, in a situation where cloud computing is mainstream, it is a general trend to use edge computing to improve the response speed of the cloud. In the case of chatbots, having a broker with a cache function on the edge has an advantage in terms of reaction speed. In the case of the knowledge coverage of the chatbot system, it is natural that a multi-chatbot with several exposed chatbots is helpful from the point of view of collective intelligence rather than a single chatbot. However, in a multi-chatbot system, if the client uses a structure in which a client asks a question to each chatbot and selects an answer, the slowdown will become more serious. The usage of the cache mechanism improves the speed of a broker.

We used the concepts of relevance and frequency in the cache mechanism in this paper. As one of our future works, we plan to use the concept of likes in addition to relevance and frequency. This idea will promote competition among chatbots for the advancement of knowledge. In addition, to make the cache mechanism more intelligent, one needs to use an ontology or knowledge graph. The knowledge graph allows the cache mechanism to consider the meaning of words based on conversation history and ontology.

Author Contributions: Conceptualization, O.M. and D.K.; methodology, O.M. and D.K.; software, O.M.; validation, D.K.; formal analysis, D.K.; investigation, O.M. and D.K.; resources, O.M. and D.K.; data curation, D.K.; writing—original draft preparation, O.M.; writing—review and editing, D.K.; visualization, O.M.; supervision, D.K.; project administration, D.K.; funding acquisition, D.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program, grant number IITP-2021-2016-0-00465, supervised by the ITTP (Institute for Information & Communications Technology Planning & Evaluation).

Data Availability Statement: Publicly available datasets were analyzed in this study. This data can be found here: https://code.google.com/archive/p/aiml-en-us-foundation-alice/downloads.

Acknowledgments: This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2021-2016-0-00465) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Makhkamova, O.; Lee, K.-H.; Do, K.; Kim, D. Deep Learning-based Multi-Chatbot Broker for Q&A Improvement of Video Tutoring Assistant. In Proceedings of the 2020 IEEE International Conference on Big Data and Smart Computing (BigComp), Busan, Korea, 19–22 February 2020; pp. 221–224.
2. Daniel, J. Making sense of MOOCs: Musings in a maze of myth, paradox and possibility. *J. Interact. Media Educ.* **2012**, *3*, 18. [CrossRef]
3. Glusac, D.; Karuović, D.; Milanov, D. Massive open online courses—Pedagogical overview. In the Proceedings of the 2015 16th International Carpathian Control Conference (ICCC), Szilvasvarad, Hungary, 27–30 May; 2015; pp. 142–146.
4. Theresa, Z.; Meinel, C. Towards personalized, diaolgue-based system supported learning for MOOCs. In Proceedings of the Learning Ideas Conference, New York, NY, USA, 14–18 June 2021.
5. Dumford, A.D.; Miller, A.L. Online learning in higher education: Exploring advantages and disadvantages for engagement. *J. Comput. High. Ed.* **2018**, *30*, 452–465. [CrossRef]
6. Park, K.; Kim, D.; Makhkamova, O.; Baatarzorig, J. *Design of Chatbot System for Internet Lecture Video*; The Korean Institute of Information Scientists and Engineers: Jeju, Korea, 2019; pp. 247–248.
7. Smith, A.J. Cache memories. *ACM Comput. Surv.* **1982**, *14*, 473–530. [CrossRef]
8. Sathiyamoorthi, V.; Bhaskaran, V.M. Web caching through modified cache replacement algorithm. In Proceedings of the 2012 International Conference on Recent Trends in Information Technology, Chennai, India, 19–21 April 2012; pp. 483–487.

9. Boyanov, M.; Koychev, I.; Nakov, P.; Moschiti, A.; Martino, G.D.S. Building chatbots from forum data: Model selection using question answering metrics. In Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP, Varna, Bulgaria, 4–6 September 2017; pp. 121–129.
10. Rana, M. EagleBot: A Chatbot Based Multi-tier Question Answering System for Retrieving Answers from Heterogeneous Sources Using BERT. Master's Thesis, Georgia Southern University, Statesboro, GA, USA, 1994.
11. Espasa, A.; Meneses, J. Analyzing feedback processes in an online teaching and learning environment: An exploratory study. *J. High. Educ.* **2010**, *59*, 277–292. [CrossRef]
12. Feng, W.; Tang, J.; Liu, T.X.; Zhang, S.; Guan, J. Understanding dropouts in MOOCs. In Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI'19), Hilton Hawaiian Village, Honolulu, HI, USA, 27 January–1 February 2019.
13. Loghin, D.; Ramapantulu, L.; Teo, Y.M. Towards Analyzing the Performance of Hybrid Edge-Cloud Processing. In Proceedings of the 2019 IEEE International Conference on Edge Computing (EDGE), Milan, Italy, 8–13 July 2019; pp. 87–94.
14. Borsci, S.; Malizia, A.; Schmettow, M.; Velde, F.; Tariverdiyeva, G.; Balaji, D.; Chamberlain, A. The Chatbot Usability Scale: The Design and Pilot of a Usability Scale for Interaction with AI-Based Conversational Agents. *Pers. Ubiquit. Comput.* **2021**, 1–25. [CrossRef]
15. Kowsher, M.; Tahabilder, A.; Sanjid, Z.I.; Prottasha, N.J.; Sarker, M.H. Knowledge-Base Optimization to Reduce the Response Time of Bangla Chatbot. In Proceedings of the 2020 Joint 9th International Conference on Informatics, Electronics & Vision (ICIEV) and 2020 4th International Conference on Imaging, Vision & Pattern Recognition (icIVPR), Kitakyushu, Japan, 26–29 August 2020; pp. 1–6.
16. Lam, M.D.; Rothberg, E.E.; Wolf, M.E. The cache performance and optimizations of blocked algorithms. *ACM SIGOPS Oper. Syst. Rev.* **1991**, *25*, 63–74. [CrossRef]
17. Meizhen, W.; Yanlei, S.; Yue, T. The design and implementation of LRU-based web cache. In Proceedings of the 2013 8th International Conference on Communications and Networking in China (CHINACOM), Guilin, China, 14–16 August 2013; pp. 400–404.
18. Gomaa, H.; Messier, G.G.; Williamson, C.; Davies, R. Estimating Instantaneous Cache Hit Ratio Using Markov Chain Analysis. *IEEE/ACM Trans. Netw.* **2012**, *21*, 1472–1483. [CrossRef]
19. Tran, A.-T.; Lakew, D.S.; Nguyen, T.-V.; Tuong, V.-D.; Truong, T.P.; Dao, N.-N.; Cho, S. Hit Ratio and Latency Optimization for Caching Systems: A Survey. In Proceedings of the 2021 International Conference on Information Networking (ICOIN), Jeju Island, Korea, 2 February 2021; pp. 577–581.
20. Shi, L.; Wei, L.; Ye, H.-Q.; Shi, Y. Measurements of Web Caching and Applications. In Proceedings of the 2006 International Conference on Machine Learning and Cybernetics, Dalian, China, 13–16 August 2006; pp. 1587–1591.
21. Ma, H.; Wang, S.; Li, M.; Li, N. Enhancing Graph-Based Keywords Extraction with Node Association. In Proceedings of the on the Move to Meaningful Internet Systems: OTM 2015 Workshops, Rhodes, Greece, 26–30 October 2015; pp. 497–510.
22. Marietto, M.D.G.B.; Aguiar, R.V.; Barbosa, G.D.O.; Botelho, W.T.; Pimentel, E.; Franca, R.D.S.; Da Silva, V.L. Artificial Intelligence Markup Language: A Brief Tutorial. *arXiv* **2013**, arXiv:1307.3091. [CrossRef]
23. Li, W.; Zhao, J. TextRank Algorithm by Exploiting Wikipedia for Short Text Keywords Extraction. In Proceedings of the 2016 3rd International Conference on Information Science and Control Engineering (ICISCE), Beijing, China, 8–10 July 2016; pp. 683–686.
24. Geetha, K.; Gounden, N.A.; Monikandan, S. SEMALRU: An Implementation of modified web cache replacement algorithm. In Proceedings of the 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC), Bhubaneswar, India, 8–10 February 2009; pp. 1406–1410. [CrossRef]
25. Google Code, AIML Foundation Archive. Available online: https://code.google.com/archive/p/aiml-en-us-foundation-alice/downloads (accessed on 3 March 2021).